QuanEstimation: An open-source toolkit for quantum parameter estimation

A tutorial

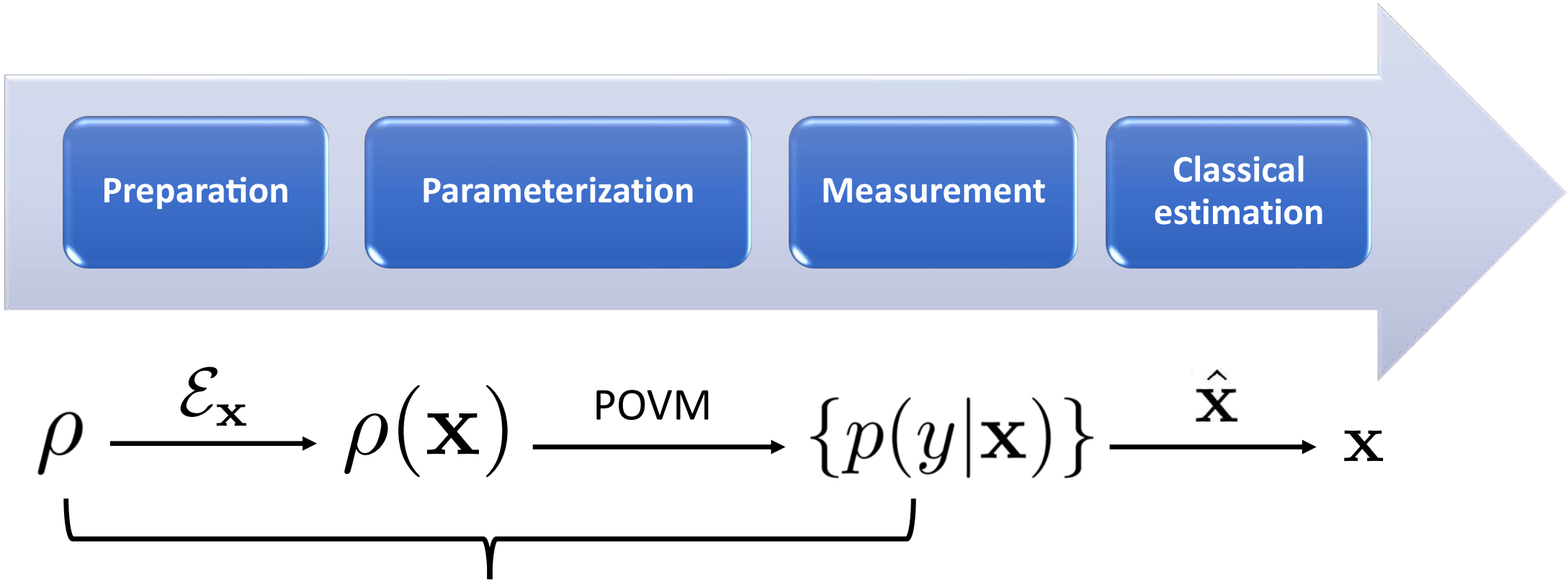
Jing Liu (刘 京)

National Precise Gravity Measurement Facility, School of Physics,
Huazhong University of Science and Technology

liujingphys@hust.edu.cn

2023@Hangzhou

Quantum parameter estimation



ppm: major concern in quantum metrology

ppm: parts per million

C. W. Helstrom, Quantum Detection and Estimation Theory (New York: Academic, 1976).

A. S. Holevo, Probabilistic and Statistical Aspects of Quantum Theory (Amsterdam: North-Holland, 1982).

J. Liu et al., J. Phys. A: Math. Theor. **53**, 023001 (2020).

Towards the design of optimal schemes

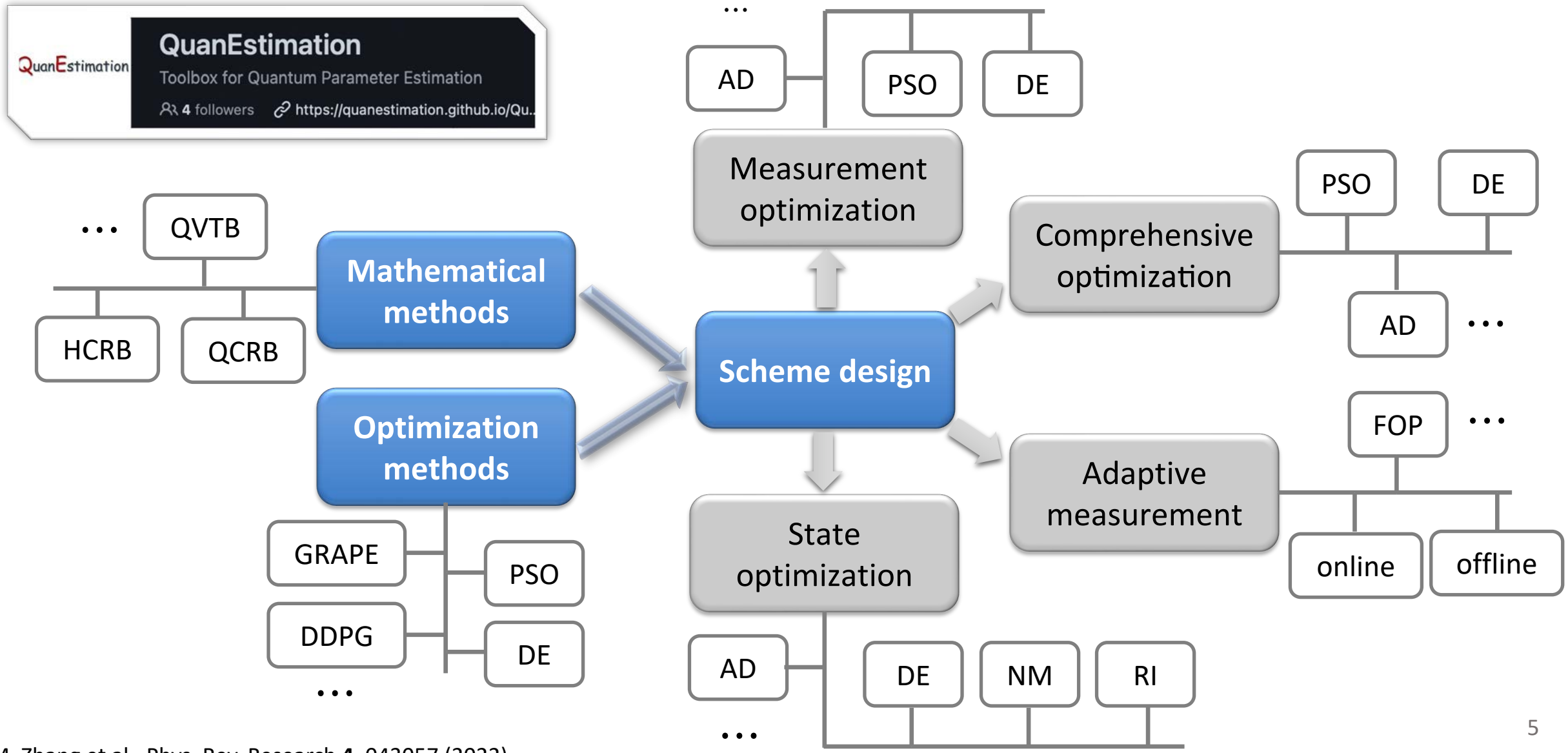
$$\rho \xrightarrow{\mathcal{E}_{\mathbf{x}}} \rho(\mathbf{x}) \xrightarrow{\text{POVM}} \{p(y|\mathbf{x})\} \xrightarrow{\hat{\mathbf{x}}} \mathbf{x}$$

In a specific scenario:

(1) Choose feasible metrological bounds or tools

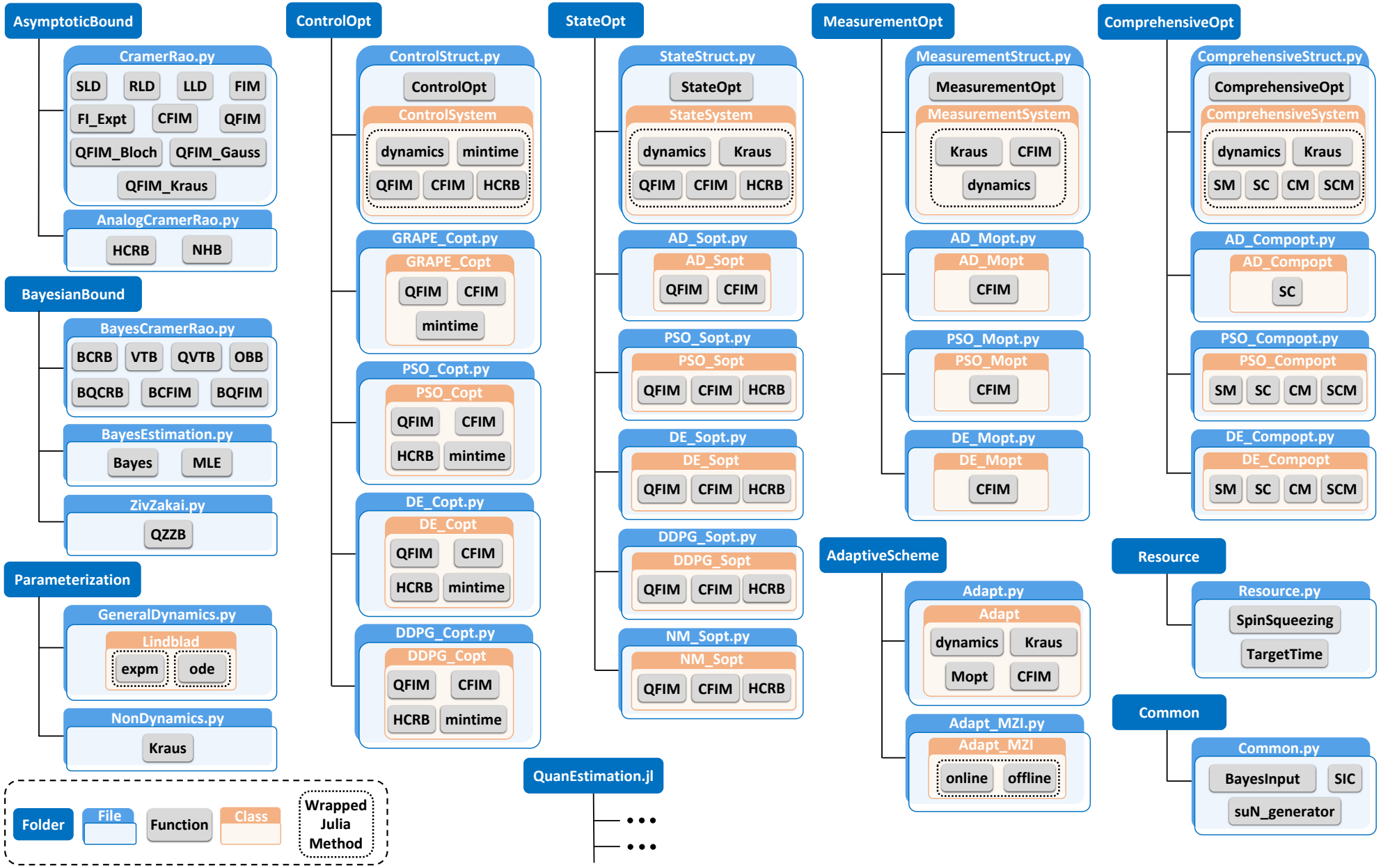
(2) Optimizations of ‘ppm’

QuanEstimation: an open-source Python-Julia package



M. Zhang et al., Phys. Rev. Research **4**, 043057 (2022).

File structure



Parameterization process

The dynamics is governed by the quantum master equation

$$\partial_t \rho = \mathcal{L}\rho = -i[H, \rho] + \sum_i \gamma_i \left(\Gamma_i \rho \Gamma_i^\dagger - \frac{1}{2}\{\rho, \Gamma_i^\dagger \Gamma_i\} \right)$$

The derivative of density matrix with respect to parameters

$$\begin{aligned} \partial_{\mathbf{x}} \rho_j &= \Delta t_j (\partial_{\mathbf{x}} \mathcal{L}) \rho_j + e^{\Delta t_j \mathcal{L}} (\partial_{\mathbf{x}} \rho_{j-1}) \\ &= -i \Delta t_j [\partial_{\mathbf{x}} H_0, \rho_j] + e^{\Delta t_j \mathcal{L}} (\partial_{\mathbf{x}} \rho_{j-1}) \end{aligned}$$

```
dynamics = Lindblad(tspan,rho0,H0,dH,decay=[],Hc=[],ctrl=[])
rho,drho = dynamics.expm()
```

Solving the ordinary differential equation

$$\partial_t (\partial_{\mathbf{x}} \rho) = -i[\partial_{\mathbf{x}} H, \rho] + \mathcal{L}(\partial_{\mathbf{x}} \rho)$$

```
rho,drho = dynamics.ode()
```

Mathematical methods

- Cramer-Rao bound

For a density matrix in which a vector of unknown parameters $\mathbf{x} = (x_0, x_1, \dots)^{\mathrm{T}}$ is encoded, the covariance matrix $\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\})$ for any POVM $\{\Pi_y\}$ and unbiased estimators $\hat{\mathbf{x}} = (\hat{x}_0, \hat{x}_1, \dots)^{\mathrm{T}}$ satisfies the following inequality

$$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geqslant \frac{1}{n}\mathcal{I}^{-1}(\{\Pi_y\}) \geqslant \frac{1}{n}\mathcal{F}^{-1}$$

Single parameter

$$\delta x_i \geq \frac{1}{\sqrt{n\mathcal{F}_{ii}}}$$

$\mathcal{I}$ Classical Fisher information matrix (CFIM)

$\mathcal{F}$ Quantum Fisher information matrix (QFIM)

n Repetition number of experiments

Mathematical methods

- Cramer-Rao bound

For a probability distribution $\{p(y|\mathbf{x})\}$

The entry of CFIM is defined as $\mathcal{I}_{ab} = \sum_y \frac{1}{p(y|\mathbf{x})}[\partial_a p(y|\mathbf{x})][\partial_b p(y|\mathbf{x})]$

```
FIM(p,dp,eps=1e-8)
```

For a parameterized density matrix ρ ➡ $\{p(y|\mathbf{x}) = \mathrm{Tr}(\rho\Pi_y)\}$

The derivatives of density matrix with respect to parameters $[\partial_0\rho, \partial_1\rho, \ldots]$

```
CFIM(rho,drho,M=[],eps=1e-8)
```

Mathematical methods

- Cramer-Rao bound

Symmetric Logarithmic Derivative

$$\langle \lambda_i | L_a | \lambda_j \rangle = \frac{2\langle \lambda_i | \partial_a \rho | \lambda_j \rangle}{\lambda_i + \lambda_j}$$

```
SLD(rho,drho,rep="original",eps=1e-8)
```

Right Logarithmic Derivative

$$\langle \lambda_i | \mathcal{R}_a | \lambda_j \rangle = \frac{1}{\lambda_i} \langle \lambda_i | \partial_a \rho | \lambda_j \rangle$$

```
RLD(rho,drho,rep="original",eps=1e-8)
```

Left Logarithmic Derivative

$$\langle \lambda_i | \mathcal{R}_a^\dagger | \lambda_j \rangle = \frac{1}{\lambda_j} \langle \lambda_i | \partial_a \rho | \lambda_j \rangle$$

```
LLD(rho,drho,rep="original",eps=1e-8)
```

Mathematical methods

- Cramer-Rao bound

The entry of QFIM is defined as $\mathcal{F}_{ab} = \frac{1}{2}\mathrm{Tr}(\rho\{L_a, L_b\})$

or $\mathcal{F}_{ab} = \mathrm{Tr}(\rho\mathcal{R}_a\mathcal{R}_b^\dagger)$

```
QFIM(rho,drho,LDtype="SLD",exportLD=False,eps=1e-8)
```

More expressions [see J. Liu et al., J. Phys. A: Math. Theor. **53** 023001 (2020).]

```
QFIM_Kraus(rho,K,dK,LDtype="SLD",exportLD=False,eps=1e-8)
```

```
QFIM_Bloch(r,dr,eps=1e-8)
```

```
QFIM_Gauss(R,dR,D,dD)
```

Mathematical methods

Hamiltonian $H = \frac{1}{2}\omega_0\sigma_3$

Dynamics

$$\partial_t\rho = -i\left[H,\rho\right] + \gamma_+\left(\sigma_+\rho\sigma_- - \frac{1}{2}\left\{\sigma_-\sigma_+,\rho\right\}\right) + \gamma_-\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\left\{\sigma_+\sigma_-,\rho\right\}\right)$$

Initial state $|+\rangle$

Measurement $\{|+\rangle\langle+|, |-\rangle\langle-|\}$

[ ]:

```
import numpy as np
from quanestimation import *

#initial state
rho0=0.5*np.array([[1., 1.], [1., 1.]])
# Hamiltonian
omega = 1.0
sigma1 = np.array([[0., 1.], [1., 0.]])
sigma2 = np.array([[0., -1.j], [1.j, 0.]])
sigma3 = np.array([[1., 0.], [0., -1.]])
H0 = 0.5 * omega * sigma3
dH = [0.5 * sigma3]
# decay
sp = np.array([[0., 1.], [0., 0.]])
sm = np.array([[0., 0.], [1., 0.]])
decay = [[sp, 0.], [sm, 0.1]]
# time for evolution
tspan = np.linspace(0., 10., 2500)
# measurement
M1 = 0.5 * np.array([[1., 1.], [1.,1.]])
M2 = 0.5 * np.array([[1., -1.], [-1., 1.]])
M = [M1, M2]
```

Mathematical methods

Hamiltonian $H = \frac{1}{2}\omega_0\sigma_3$

Dynamics

$$\partial_t\rho = -i\left[H,\rho\right] + \gamma_+\left(\sigma_+\rho\sigma_- - \frac{1}{2}\left\{\sigma_-\sigma_+,\rho\right\}\right) + \gamma_-\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\left\{\sigma_+\sigma_-,\rho\right\}\right)$$

Initial state $|+\rangle$

Measurement $\{|+\rangle\langle+|, |-\rangle\langle-|\}$

[ ]:

```
import numpy as np
from quanestimation import *
from qutip import *

# initial state
rho0 = 0.5 * np.ones((2, 2))
# Hamiltonian
omega = 1.0
H0 = 0.5 * omega * sigmaz()
dH = [0.5 * sigmaz().full()]
# decay
decay = [[sigmap(), 0.], [sigmam(), 0.1]]
# time for evolution
tspan = np.linspace(0., 10., 2500)
# measurement
M1 = 0.5 * np.ones((2, 2))
M2 = identity(2) - M1
M = [M1, M2]
```

Mathematical methods

Hamiltonian $H = \frac{1}{2}\omega_0\sigma_3$

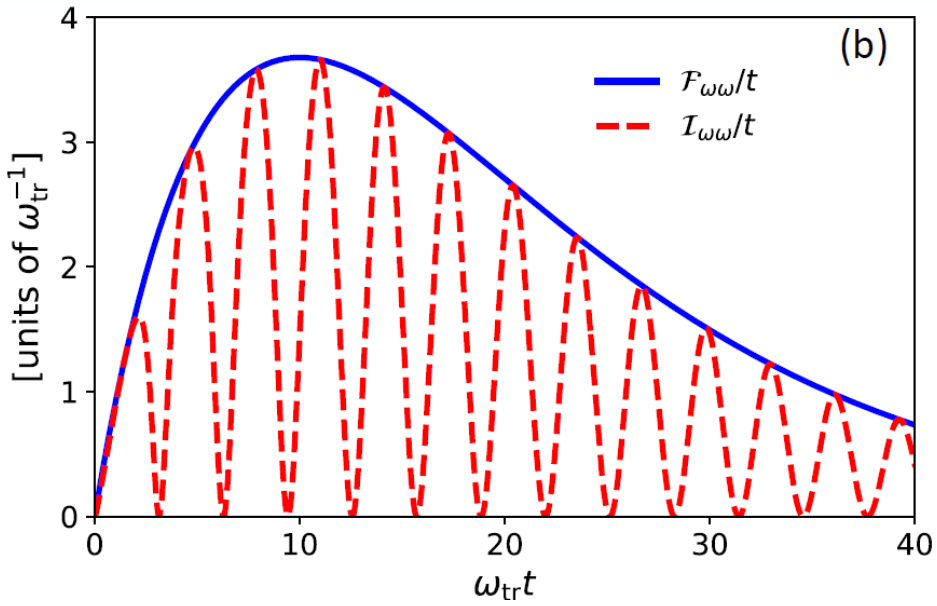
Dynamics

$$\partial_t\rho = -i\left[H,\rho\right] + \gamma_+\left(\sigma_+\rho\sigma_- - \frac{1}{2}\left\{\sigma_-\sigma_+,\rho\right\}\right)$$
$$+\gamma_-\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\left\{\sigma_+\sigma_-,\rho\right\}\right)$$

Initial state $|+\rangle$

Measurement $\{|+\rangle\langle+|, |-\rangle\langle-|\}$

```
[ ]: # parameterization
dynamics = Lindblad(tspan, rho0, H0, dH, decay)
rho, drho = dynamics.expm()
# calculation of the QFI
CFI, QFI = [], []
for ti in range(1,len(tspan)):
    CFI_tp = CFIM(rho[ti], drho[ti], M)
    QFI_tp = QFIM(rho[ti], drho[ti])
    CFI.append(CFI_tp)
    QFI.append(QFI_tp)
```



Mathematical methods

- Holevo Cramer-Rao bound

$$\mathrm{Tr}(W\,\mathrm{cov}(\hat{\mathbf{x}},\{\Pi_y\})) \geq \min_{\mathbf{X},V} \mathrm{Tr}(WV)$$

$$\min_{\mathbf{X},V}\ \mathrm{Tr}(WV),$$

$$\text{subject to } \begin{cases} \begin{pmatrix} V & \Lambda^{\mathrm{T}} R^{\dagger} \\ R\Lambda & \mathbb{1} \end{pmatrix} \geq 0, \\ \sum_i [\Lambda]_{ai}\,\mathrm{Tr}(\lambda_i \partial_b \rho) = \delta_{ab}. \end{cases}$$

$$V \geq Z(\mathbf{X})$$

$$\mathbf{X} = [X_0, X_1, \cdots]$$

$$Z = \Lambda^{\mathrm{T}} R^{\dagger} R \Lambda$$

$$X_i = \textstyle\sum_j [\Lambda]_{ij} \lambda_j$$

$$[Z(\mathbf{X})]_{ab} := \mathrm{Tr}(\rho X_a X_b)$$

$$X_a := \textstyle\sum_y (\hat{x}_a(y) - x_a)\Pi_y$$

Semidefinite programming

```
HCRB(rho,drho,W,eps=1e-8)
```

A. S. Holevo, J. Multivariate Anal. **3**, 337-394 (1973).

Mathematical methods

- Nagaoka-Hayashi bound

$$\mathrm{Tr}(W\,\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\})) \geqslant \min_{\mathbf{X},\mathcal{Q}} \mathrm{Tr}((W \otimes \rho)\mathcal{Q})$$

$$\min_{\mathbf{X},\mathcal{Q}} \ \mathrm{Tr}((W \otimes \rho)\mathcal{Q}),$$

$$\text{subject to} \begin{cases} \begin{pmatrix} \mathcal{Q} & \mathbf{X}^{\mathrm{T}} \\ \mathbf{X} & \mathbb{1} \end{pmatrix} \geqslant 0, \\ \mathrm{Tr}(\rho X_a) = 0, \ \forall a, \\ \mathrm{Tr}(X_a \partial_b \rho) = \delta_{ab}, \ \forall a, b. \end{cases}$$

$$\mathcal{Q} \geqslant \mathbf{X}^{\mathrm{T}}\mathbf{X}$$

Semidefinite programming

```
NHB(rho,drho,W)
```

H. Nagaoka, Asymptotic Theory of Quantum Statistical Inference: Selected Papers (World Scientific, Singapore, 2005).

M. Hayashi, Asymptotic Theory of Quantum Statistical Inference: Selected Papers (World Scientific, Singapore, 2005).

Mathematical methods

Hamiltonian

$$H = \omega_1 \sigma_3^{(1)} + \omega_2 \sigma_3^{(2)} + g \sigma_1^{(1)} \sigma_1^{(2)}$$

Dynamics

$$\partial_t \rho = -i[H, \rho] + \sum_{i=1,2} \gamma_i \left(\sigma_3^{(i)} \rho \sigma_3^{(i)} - \rho \right)$$

Initial state $\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle)$

Measurement $\{\Pi_1, \Pi_2, I - \Pi_1 - \Pi_2\}$

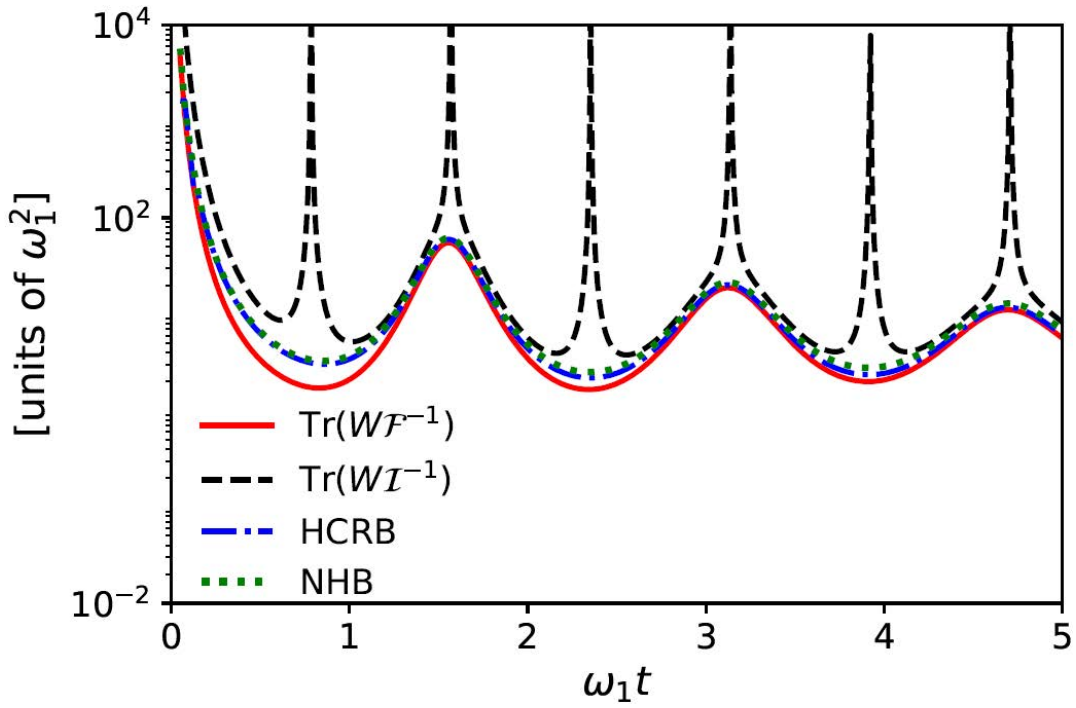
$\Pi_1 = 0.85|00\rangle\langle 00| \quad \Pi_2 = 0.1|++\rangle\langle ++|$

[ ]:

```
import numpy as np
from quanestimation import *
from qutip import *

# initial state
psi0 = np.array([1., 0., 0., 1.])/np.sqrt(2)
rho0 = np.dot(psi0.reshape(-1, 1), psi0.reshape(1, -1).conj())

# free Hamiltonian
omega1, omega2, g = 1.0, 1.0, 0.1
s1, s2, s3 = sigmax(), sigmay(), sigmaz()
I2 = identity(2)
H0 = omega1*np.kron(s3, I2) + omega2*np.kron(I2, s3)
                            + g*np.kron(s1, s1)

# derivatives of the free Hamiltonian on omega2 and g
dH = [np.kron(I2, s3), np.kron(s1, s1)]

# dissipation
decay = [[np.kron(s3, I2), 0.05], [np.kron(I2, s3), 0.05]]

# measurement
m1 = np.array([1., 0., 0., 0.])
M1 = 0.85*np.dot(m1.reshape(-1, 1), m1.reshape(1, -1).conj())
M2 = 0.1*np.ones((4, 4))
M = [M1, M2, np.identity(4) - M1 - M2]
```

Mathematical methods

```
[ ]: # weight matrix
     W = np.identity(2)
     # time length for the evolution
     tspan = np.linspace(0., 5., 200)
     # dynamics
     dynamics = Lindblad(tspan, rho0, H0, dH, decay)
     rho, drho = dynamics.expm()
     # calculation of the HCRB and NHB
     HCRB_value, NHB_value = [], []
     for ti in range(1, len(tspan)):
         HCRB_tp = HCRB(rho[ti], drho[ti], W, eps=1e-7)
         HCRB_value.append(HCRB_tp)
         NHB_tp = NHB(rho[ti], drho[ti], W)
         NHB_value.append(NHB_tp)
```



Mathematical methods

- Bayesian estimation

Bayes' rule $p(\mathbf{x}|y) = \dfrac{p(y|\mathbf{x})p(\mathbf{x})}{\int p(y|\mathbf{x})p(\mathbf{x})d\mathbf{x}}$

Mean value estimation $\hat{\mathbf{x}} = \int \mathbf{x}p(\mathbf{x}|y)d\mathbf{x}$

```
pout,xout = Bayes(x,p,rho,y,M=[],estimator="mean",
                  savefile=False)
```

Maximum a posteriori estimation $\hat{\mathbf{x}} = \text{argmax}_{\mathbf{x}}\, p(\mathbf{x}|y)$ → `estimator="MAP"`

The maximum likelihood estimation $\hat{\mathbf{x}} = \text{argmax}_{\mathbf{x}} \prod_i p(y_i|\mathbf{x})$

```
Lout,xout = MLE(x,rho,y,M=[],savefile=False)
```

Mathematical methods

Hamiltonian $H = \frac{\kappa\omega_0}{2}(\sigma_1 \cos x + \sigma_3 \sin x)$ Dynamics $\partial_t \rho = -i[H, \rho]$

Measurement $\{\Pi_1, \Pi_2, I - \Pi_1 - \Pi_2\}$ Initial state $|+\rangle$

$\Pi_1 = 0.85|00\rangle\langle 00| \; \Pi_2 = 0.1|++\rangle\langle ++|$

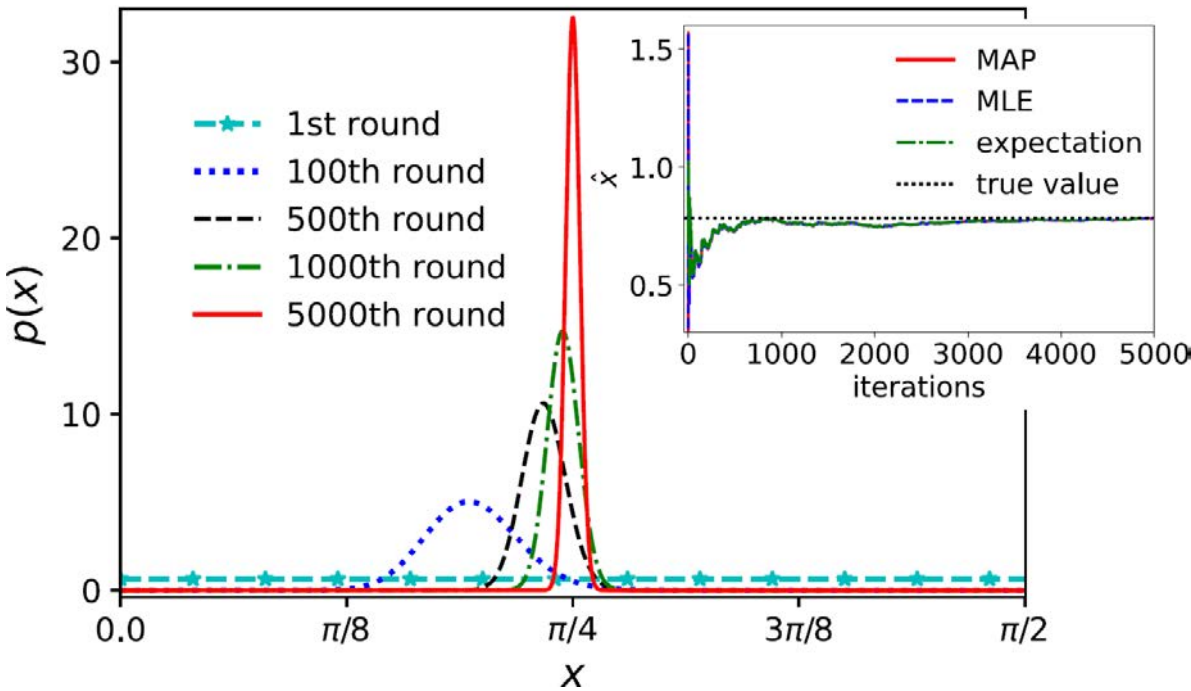
[ ]:
```
from quanestimation import *
from qutip import *
import numpy as np
import random

# initial state
rho0 = 0.5 * np.ones((2, 2))
# free Hamiltonian
kappa, omega0 = np.pi/2.0, 1.0
s1, s2, s3 = sigmax(), sigmay(), sigmaz()
s1, s2, s3 = s1.full(), s2.full(), s3.full()
H0_func = lambda x: 0.5*kappa*omega0*(s1*np.cos(x)+s3*np.sin(x))
# derivative of the free Hamiltonian on x
dH_func = lambda x: [
    0.5*kappa*omega0*(-s1*np.sin(x)+s3*np.cos(x))
]
```

[ ]:
```
# time span for the evolution
tspan = np.linspace(0., 1., 1000)
# parameterization
x = np.linspace(0., 0.5*np.pi, 1000)
rho = [
    np.zeros((len(rho0), len(rho0)), dtype=np.complex128)
    for i in range(len(x))
]
for i in range(len(x)):
    H0 = H0_func(x[i])
    dH = dH_func(x[i])
    dynamics = Lindblad(tspan, rho0, H0, dH)
    rho_tp, drho_tp = dynamics.expm()
    rho[i] = rho_tp[-1]

# measurement
M1 = 0.5 * np.ones((2, 2))
M2 = (identity(2) - M1).full()
M = [M1, M2]
```

Mathematical methods

```
[ ]: # prior distribution
     p = (1.0/(x[-1]-x[0]))*np.ones(len(x))
     # Generation of the experimental results
     y = [ 0. if np.random.randn() > 0.5 else 1. for _ in range(500)]
```

```
[ ]: # mean value estimation
     pout, xout = Bayes([x], p, rho, y, M=M, estimator="mean",
                        savefile=True)
     # maximum a posteriori estimation
     pout, xout = Bayes([x], p, rho, y, M=M, estimator="MAP",
                        savefile=True)
     # maximum likelihood estimation
     Lout, xout = MLE([x], rho, y, M=M, savefile=True)
```



Mathematical methods

- Bayesian cost

Bayesian cost for the quadratic cost

$$\bar{C} := \int p(\mathbf{x}) \sum_y p(y|\mathbf{x})(\mathbf{x} - \hat{\mathbf{x}})^{\mathrm{T}} W (\mathbf{x} - \hat{\mathbf{x}})\, d\mathbf{x}$$

```
BayesCost(x,p,xest,rho,M,W=[],eps=1e-8)
```

Average Bayesian cost bound

$$\bar{C} \geqslant \int p(\mathbf{x})(\mathbf{x}^{\mathrm{T}} W \mathbf{x}) d\mathbf{x} - \sum_{ab} W_{ab} \mathrm{Tr}(\bar{\rho} \bar{L}_a \bar{L}_b)$$

```
BCB(x,p,rho,W=[],eps=1e-8)
```

Mathematical methods

- Bayesian Cramer-Rao bounds

Covariance matrix $\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) = \int p(\mathbf{x}) \sum_y \mathrm{Tr}(\rho\Pi_y)(\hat{\mathbf{x}}-\mathbf{x})(\hat{\mathbf{x}}-\mathbf{x})^{\mathrm{T}} \mathrm{d}\mathbf{x}$

Van-Trees bound $\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq (\mathcal{I}_{\mathrm{prior}} + \mathcal{I}_{\mathrm{Bayes}})^{-1}$ $\mathcal{I}_{\mathrm{prior}} = \int p(\mathbf{x}) \mathcal{I}_p \mathrm{d}\mathbf{x}$

```
VTB(x,p,dp,rho,drho,M=[],eps=1e-8)
```

Quantum Van-Trees bound $\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq (\mathcal{I}_{\mathrm{prior}} + \mathcal{F}_{\mathrm{Bayes}})^{-1}$

```
QVTB(x,p,dp,rho,drho,LDtype="SLD",eps=1e-8)
```

H. L. Van Trees, Detection, estimation, and modulation theory: Part I (Wiley, New York, 1968).

M. Tsang, H. M. Wiseman, and C. M. Caves, Phys. Rev. Lett. **106**, 090401 (2011).

Mathematical methods

- Bayesian Cramer-Rao bounds

Type 1
$$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \int p(\mathbf{x}) \left(B\mathcal{I}^{-1}B + \mathbf{b}\mathbf{b}^{\mathrm{T}} \right) \mathrm{d}\mathbf{x}$$

Type 2
$$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \mathcal{B}\,\mathcal{I}_{\mathrm{Bayes}}^{-1}\,\mathcal{B} + \int p(\mathbf{x})\mathbf{b}\mathbf{b}^{\mathrm{T}}\mathrm{d}\mathbf{x}$$

Type 3
$$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \int p(\mathbf{x})\mathcal{G}\left(\mathcal{I}_p + \mathcal{I}\right)^{-1}\mathcal{G}^{\mathrm{T}}\mathrm{d}\mathbf{x}$$

```
BCRB(x,p,dp,rho,drho,M=[],b=[],db=[],btype=1,eps=1e-8)
```

$\mathbf{b} = (b(x_0), b(x_1), \dots)^{\mathrm{T}}$ $\mathbf{b}' := (\partial_0 b(x_0), \partial_1 b(x_1), \dots)^{\mathrm{T}}$ $B_{ii} = 1 + [\mathbf{b}']_i$ $\mathcal{B} = \int p(\mathbf{x}) B \mathrm{d}\mathbf{x}$

$\mathcal{I}_{\mathrm{Bayes}} = \int p(\mathbf{x})\mathcal{I}\mathrm{d}\mathbf{x}$ $[\mathcal{I}_p]_{ab} := [\partial_a \ln p(\mathbf{x})][\partial_b \ln p(\mathbf{x})]$ $\mathcal{G}_{ab} := [\partial_b \ln p(\mathbf{x})][\mathbf{b}]_a + B_{aa}\delta_{ab}$

Mathematical methods

- Bayesian Cramer-Rao bounds

Type 1 $$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \int p(\mathbf{x}) \left(B \mathcal{F}^{-1} B + \mathbf{b}\mathbf{b}^{\mathrm{T}} \right) \mathrm{d}\mathbf{x}$$

Type 2 $$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \mathcal{B}\, \mathcal{F}_{\mathrm{Bayes}}^{-1}\, \mathcal{B} + \int p(\mathbf{x}) \mathbf{b}\mathbf{b}^{\mathrm{T}} \mathrm{d}\mathbf{x} \qquad \mathcal{F}_{\mathrm{Bayes}} = \int p(\mathbf{x}) \mathcal{F} \mathrm{d}\mathbf{x}$$

Type 3 $$\mathrm{cov}(\hat{\mathbf{x}}, \{\Pi_y\}) \geq \int p(\mathbf{x}) \mathcal{G} \left(\mathcal{I}_p + \mathcal{F} \right)^{-1} \mathcal{G}^{\mathrm{T}} \mathrm{d}\mathbf{x}$$

```
BQCRB(x,p,dp,rho,drho,b=[],db=[],btypr=1,
      Ldtypt="SLD",eps=1e-8)
```

Mathematical methods

- Ziv-Zakai error bound

$$\mathrm{var}(\hat{x}, \{\Pi_y\}) \geq \frac{1}{2} \int_0^\infty \mathrm{d}\tau \tau \mathcal{V} \int_{-\infty}^\infty \mathrm{d}x \min\{p(x), p(x+\tau)\} \times \left(1 - \frac{1}{2}||\rho(x) - \rho(x+\tau)||\right)$$

$\mathcal{V}$ is the " valley-filling" operator

If the prior distribution is in a finite regime $[\alpha, \beta]$

$$\mathrm{var}(\hat{x}, \{\Pi_y\}) \geq \frac{1}{2} \int_0^{\beta-\alpha} \mathrm{d}\tau \tau \mathcal{V} \int_\alpha^\beta \mathrm{d}x \min\{p(x), p(x+\tau)\} \times \left(1 - \frac{1}{2}||\rho(x) - \rho(x+\tau)||\right)$$

```
QZZB(x,p,rho,eps=1e-8)
```

J. Ziv and M. Zakai, IEEE Trans. Inform. Theor. **15**, 386 (1969).

M. Tsang, Phys. Rev. Lett. **108**, 230401 (2012).

Mathematical methods

Hamiltonian $H = \frac{\kappa\omega_0}{2}(\sigma_1 \cos x + \sigma_3 \sin x)$ Dynamics $\partial_t \rho = -i[H, \rho]$

Initial state $|+\rangle$ Prior distribution $p(x) = \frac{1}{c\eta\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\eta^2}}$

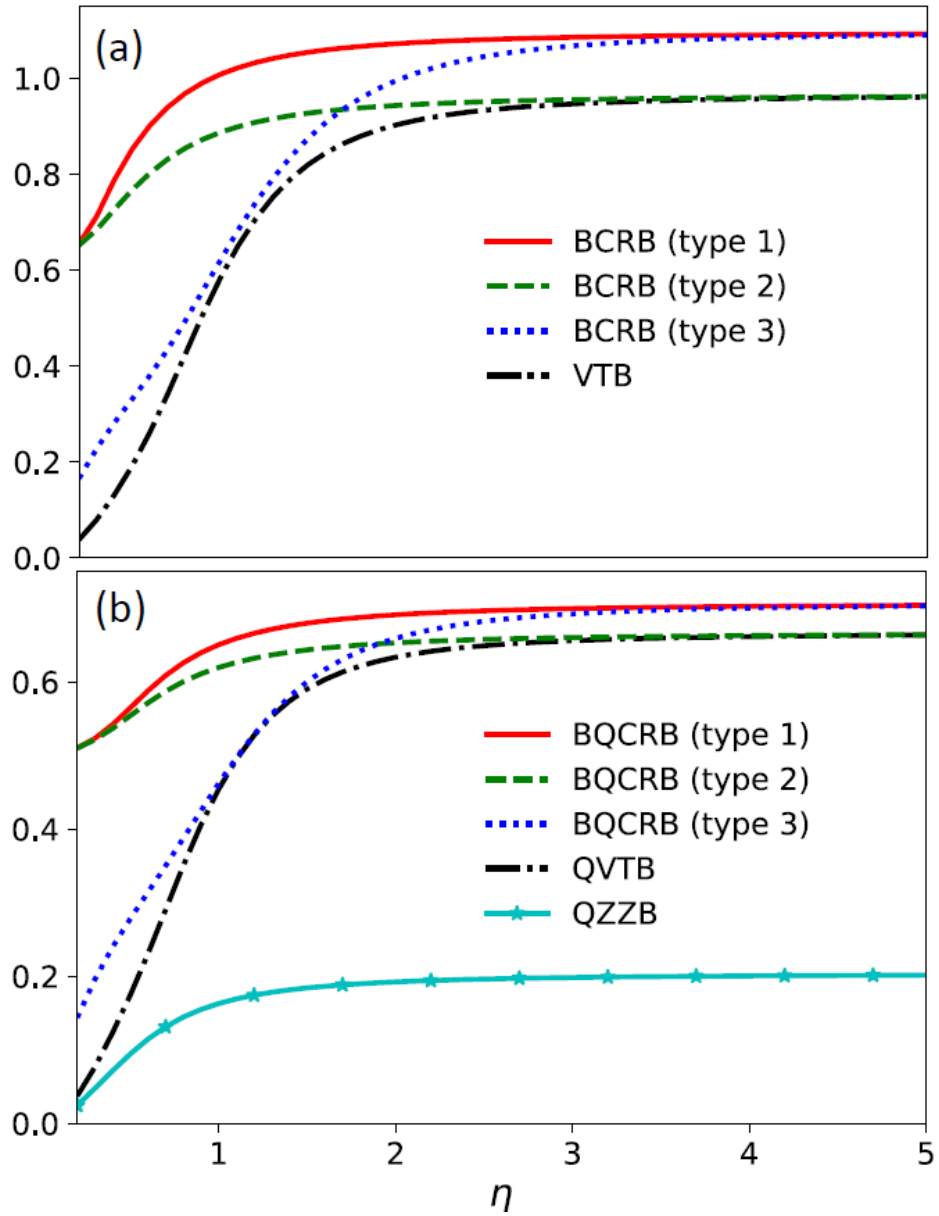
[ ]:
```
from quanestimation import *
from qutip import *
import numpy as np
import random
from scipy.integrate import simps

# initial state
rho0 = 0.5 * np.ones((2, 2))
# free Hamiltonian
kappa, omega0 = np.pi/2.0, 1.0
s1, s2, s3 = sigmax(), sigmay(), sigmaz()
s1, s2, s3 = s1.full(), s2.full(), s3.full()
H0_func = lambda x: 0.5*kappa*omega0*(s1*np.cos(x)+s3*np.sin(x))
# derivative of the free Hamiltonian on x
dH_func = lambda x: [
    0.5*kappa*omega0*(-s1*np.sin(x)+s3*np.cos(x))
]
```

[ ]:
```
# time span for the evolution
tspan = np.linspace(0., 1., 1000)
# parameterization
x = np.linspace(0., 0.5*np.pi, 1000)
dim = len(rho0)
rho = [
    np.zeros((dim, dim), dtype=np.complex128)
    for _ in range(len(x))
]
drho = [
    [np.zeros((dim, dim), dtype=np.complex128)]
    for i_ in range(len (x))
]
for i in range(len(x)):
    H0 = H0_func(x[i])
    dH = dH_func(x[i])
    dynamics = Lindblad(tspan, rho0, H0, dH)
    rho_tp, drho_tp = dynamics.expm()
    rho[i] = rho_tp[-1]
    drho[i] = drho_tp[-1]
```

Mathematical methods

```
[ ]: # prior distribution
mu, eta = 0.0, 0.2
p_func = lambda x, mu, eta: np.exp(-(x-mu) **2/(2*eta**2))\
                            /(eta*np.sqrt(2*np.pi))
dp_func = lambda x, mu, eta: -(x-mu)*np.exp(-(x-mu)**2/\
                             (2*eta**2))/(eta**3*np.sqrt (2*np.pi))
p_tp = [p_func(x[i], mu, eta) for i in range(len(x))]
dp_tp = [dp_func(x[i], mu, eta) for i in range(len(x))]
c = simps(p_tp, x)
p, dp = p_tp/c, dp_tp/c
```

```
[ ]: # Bayesian Cramer-Rao bound
f_BCRB1 = BCRB([x], p, [], rho, drho, M=[], btype=1)
f_BCRB2 = BCRB([x], p, [], rho, drho, M=[], btype=2)
f_BCRB3 = BCRB([x], p, dp, rho, drho, M=[], btype=3)
f_VTB = VTB([x], p, dp, rho, drho, M=[])
# Bayesian quantum Cramer-Rao bound
f_BQCRB1 = BQCRB([x], p, [], rho, drho, btype =1)
f_BQCRB2 = BQCRB([x], p, [], rho, drho, btype=2)
f_BQCRB3 = BQCRB([x], p, dp, rho, drho, btype =3)
f_QVTB = QVTB([x], p, dp, rho, drho)
f_QZZB = QZZB([x], p, rho)
```

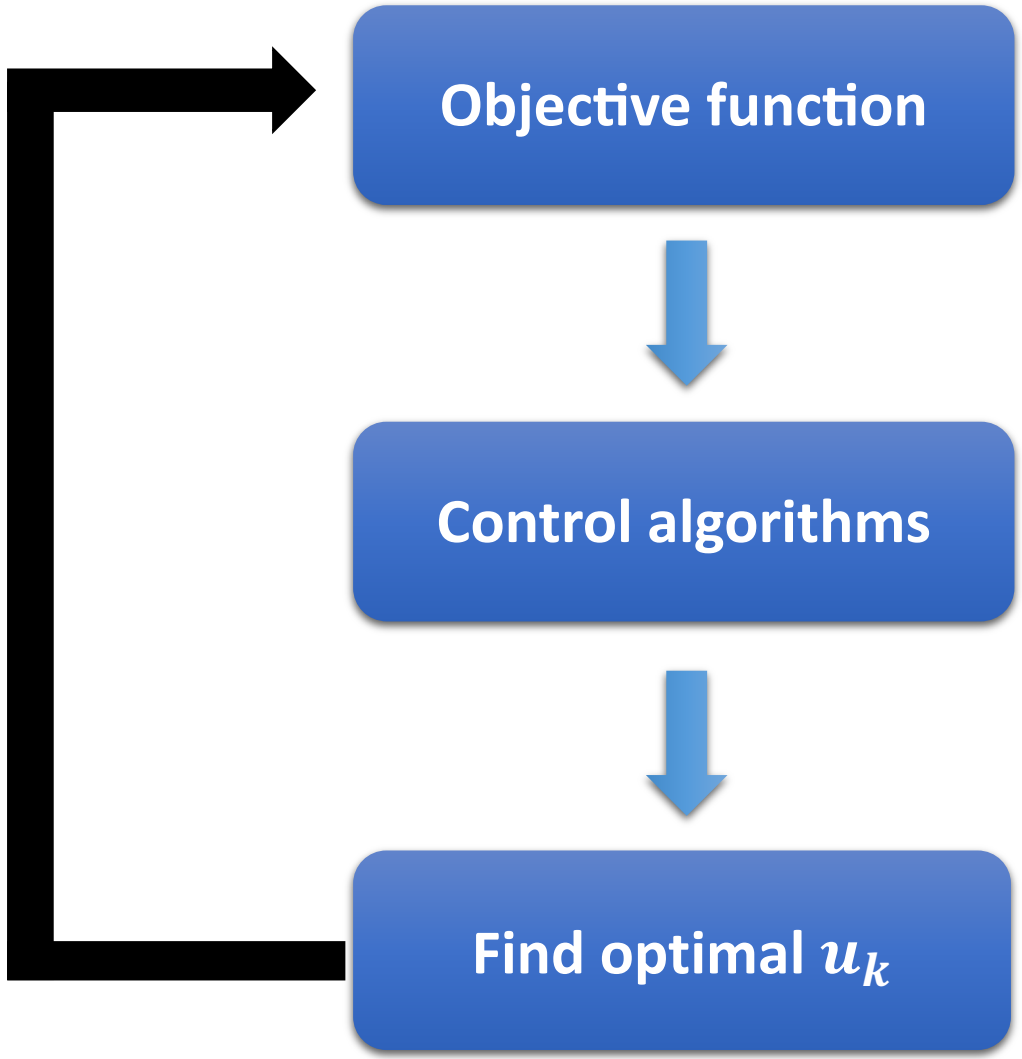


Optimizations in the parameterization process

Control

$$H = H_0(\mathbf{x}) + \sum_{k=1}^{K} u_k(t) H_k$$

Quantum control



maximize

or

minimize

CFI (CFIM), QFI (QFIM),
quantum resources ……

GRAPE, PSO, DE …

$$H = H_0(\mathbf{x}) + \sum_{k=1}^{K} u_k(t) H_k$$

Quantum control

Hamiltonian $H = H_0(\mathbf{x}) + \sum_{k=1}^{K} u_k(t) H_k$

Dynamics $\partial_t \rho = -i[H, \rho] + \sum_i \gamma_i \left(\Gamma_i \rho \Gamma_i^\dagger - \frac{1}{2} \left\{ \rho, L_i^\dagger L_i \right\} \right)$

Objective function $\begin{cases} \mathcal{F} \text{ or } \mathcal{I} \\ 1/\mathrm{Tr}(W\mathcal{F}^{-1}) \text{ or } 1/\mathrm{Tr}(W\mathcal{I}^{-1}) \end{cases}$

```
control = ControlOpt(savefile=False, method="auto-GRAPE",
                     **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay= [],
                 ctr1_bound=[], dyn_method="expm")
control.QFIM(W=[], LDtype="SLD")
control.CFIM(M=[], W=[])
control.HCRB(W=[])
```

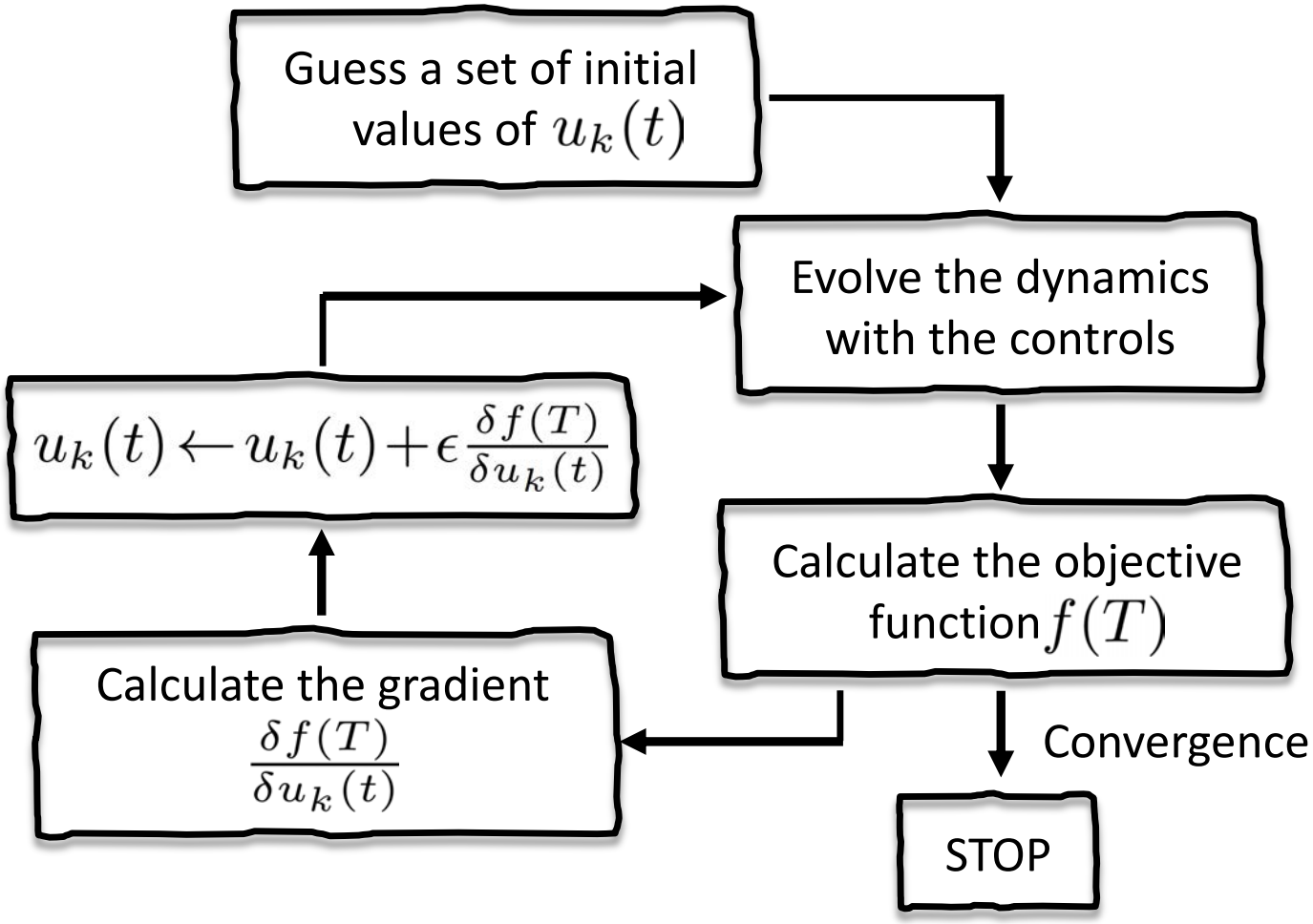
Quantum control

Control optimization algorithms in QuanEstimation

Algorithms	method=	**kwargs and default values	
auto-GRAPE (GRAPE)	"auto-GRAPE" ("GRAPE")	"Adam"	True
		"ctrl0"	[]
		"max_episode"	300
		"epsilon"	0.01
		"beta1"	0.90
		"beta2"	0.99
PSO	"PSO"	"p_num"	10
		"ctrl0"	[]
		"max_episode"	[1000,100]
		"c0"	1.0
		"c1"	2.0
		"c2"	2.0
		"seed"	1234

Algorithms	method=	**kwargs and default values	
DE	"DE"	"p_num"	10
		"ctrl0"	[]
		"max_episode"	1000
		"c"	1.0
		"cr"	0.5
		"seed"	1234
DDPG	"DDPG"	"ctrl0"	[]
		"max_episode"	500
		"layer_num"	3
		"layer_dim"	200
		"seed"	1234

Gradient ascent pulse engineering



Analytical gradient

Automatic gradient

auto-GRAPE

N. Khaneja et al., J. Magn. Reson. **172**, 296 (2005).

J. Liu and H. Yuan, Phys. Rev. A **96**, 012117 (2017); **96**, 042114 (2017).

M. Zhang et al., Phys. Rev. Research **4**, 043057 (2022).

What is automatic differentiation (AD) ?

✗ Not finite differences

$$\Delta_h f(x) = f(x+h) - f(x)$$

✗ Not symbolic differentiations

or “Computer algebra system”

✓ Chain rule of composite functions

$$y = f(g(h(x)))$$

$$\omega_0 = x$$
$$\omega_1 = h(\omega_0)$$
$$\omega_2 = g(\omega_1)$$
$$\omega_3 = f(\omega_2) = y$$

$$\frac{\partial y}{\partial x} = \frac{\partial f(\omega_2)}{\partial \omega_2} \frac{\partial g(\omega_1)}{\partial \omega_1} \frac{\partial h(\omega_0)}{\partial \omega_0}$$

Forward mode AD

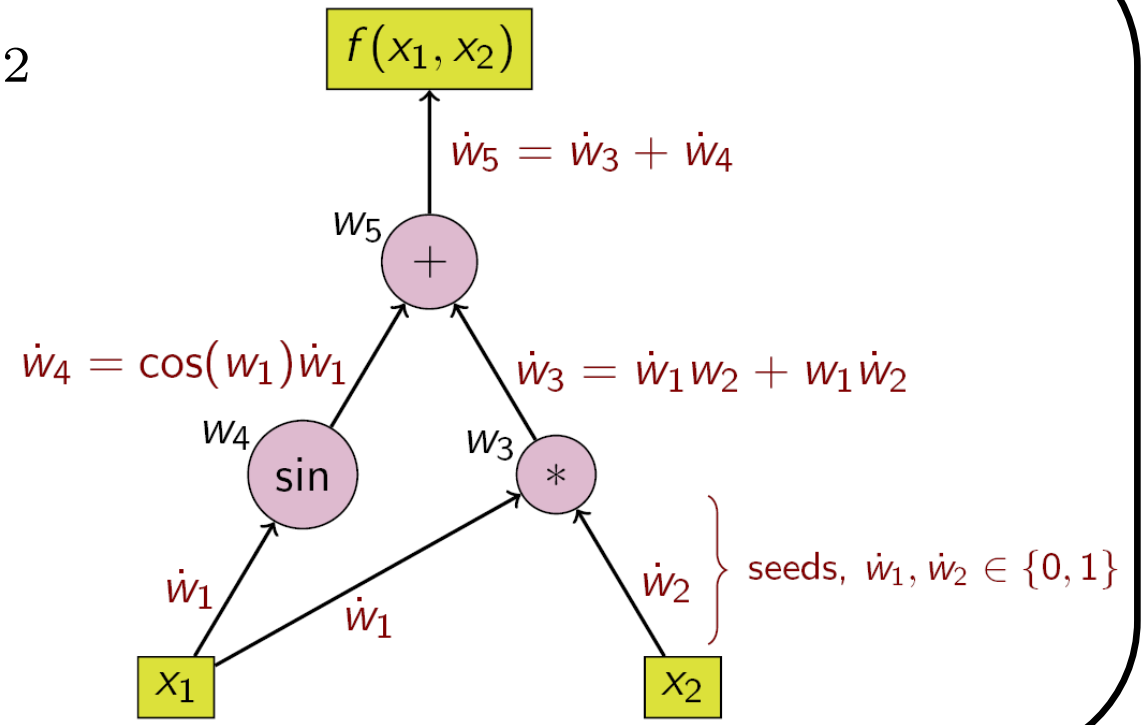
Forward accumulation

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \frac{\partial w_{n-1}}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \frac{\partial w_{n-2}}{\partial x} \right) = \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \left(\frac{\partial w_{n-2}}{\partial w_{n-3}} \frac{\partial w_{n-3}}{\partial x} \right) \right) = \cdots$$

E. g. $f(x_1, x_2) = \sin(x_1) + x_1 x_2$

Operations to compute derivative

$$\dot{w}_1 = 1 \text{ (seed)}$$
$$\dot{w}_2 = 0 \text{ (seed)}$$
$$\dot{w}_3 = w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2$$
$$\dot{w}_4 = \cos w_1 \cdot \dot{w}_1$$
$$\dot{w}_5 = 1 \cdot \dot{w}_3 + 1 \cdot \dot{w}_4$$



Reverse mode AD

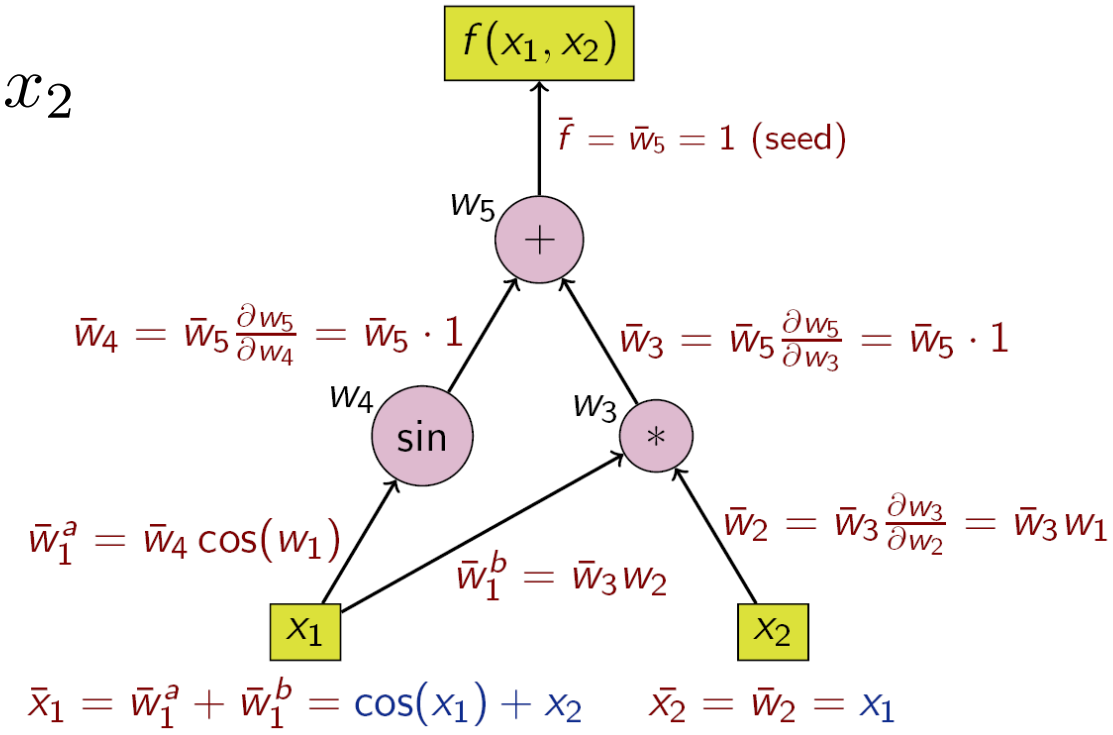
Reverse accumulation

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x}$$

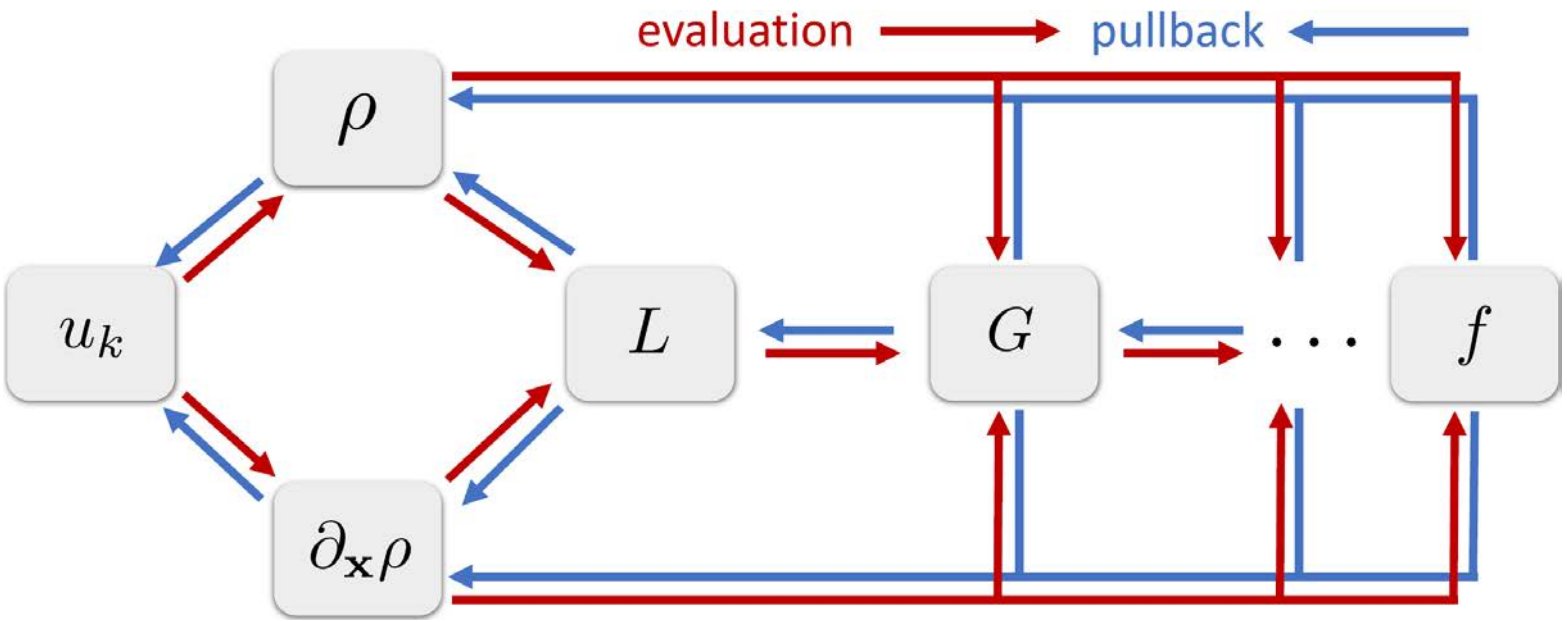
E. g. $f(x_1, x_2) = \sin(x_1) + x_1 x_2$

Operations to compute derivative

$$\bar{w}_5 = 1(\text{seed})$$
$$\bar{w}_4 = \bar{w}_5 \cdot 1$$
$$\bar{w}_3 = \bar{w}_5 \cdot 1$$
$$\bar{w}_2 = \bar{w}_3 \cdot w_1$$
$$\bar{w}_1 = \bar{w}_3 \cdot w_2 + \bar{w}_4 \cdot \cos w_1$$



AD for Quantum Fisher information matrix



$$\frac{\partial G_{\alpha\beta}}{\partial \rho} = 2h,$$

$$\frac{\partial G_{\alpha\beta}}{\partial(\partial_{\mathbf{x}}\rho)} = -hL^{\mathrm{T}} - L^{\mathrm{T}}h$$

Gradient ascent pulse engineering

Hamiltonian $H = \frac{1}{2}\omega_0\sigma_3 + \sum_{i=1}^{3} u_i(t)\sigma_i$

Dynamics

$$\partial_t\rho = -i\left[H,\rho\right] + \gamma_+\left(\sigma_+\rho\sigma_- - \frac{1}{2}\left\{\sigma_-\sigma_+,\rho\right\}\right) + \gamma_-\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\left\{\sigma_+\sigma_-,\rho\right\}\right)$$

Initial state $|+\rangle$

Measurement $\{|+\rangle\langle+|, |-\rangle\langle-|\}$

[ ]:
```
import numpy as np
from quanestimation import *
from qutip import *
# initial state
rho0 = 0.5 * np.ones((2, 2))
# Hamiltonian
omega = 1.0
s1, s2, s3 = sigmax(), sigmay(), sigmaz()
s1, s2, s3 = s1.full(), s2.full(), s3.full()
H0 = 0.5* omega * s3
dH = [0.5 * s3]
# decay
sp = destroy(2).full()
sm = create(2).full()
decay = [[sp, 0.], [sm, 0.1]]
# time for evolution
tspan = np. linspace (0., 10., 2500)
# measurement
M1 = 0.5 * np.ones((2, 2))
M2 = identity(2) - M1
M = [M1, M2]
```

[ ]:
```
# initial guess
cnum = len(tspan) -1
ctrl0 = [np.array([np.zeros(cnum) for _ in range(len(Hc))])]
# control Hamiltonian
Hc = [s1, s2, s3]
```

Gradient ascent pulse engineering

GRAPE

```
kwargs = {"Adam": True, "ctrl0":ctrl0, "max_episode": 300, \
          "epsilon": 0.01, "beta1": 0.90, "beta2": 0.99}
control = ControlOpt(savefile=False, method = "GRAPE", **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay, \
                 ctrl_bound=[-2.0, 2.0], dyn_method="expm")
control.QFIM()
```

auto-GRAPE

```
kwargs = {"Adam": True, "ctrl0":ctrl0, "max_episode": 300, \
          "epsilon": 0.01, "beta1": 0.90, "beta2": 0.99}
control = ControlOpt(savefile=False, method = "auto-GRAPE",\
                     **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay, \
                 ctrl_bound=[-2.0, 2.0], dyn_method="expm")
control.QFIM()
```

Gradient ascent pulse engineering

GRAPE

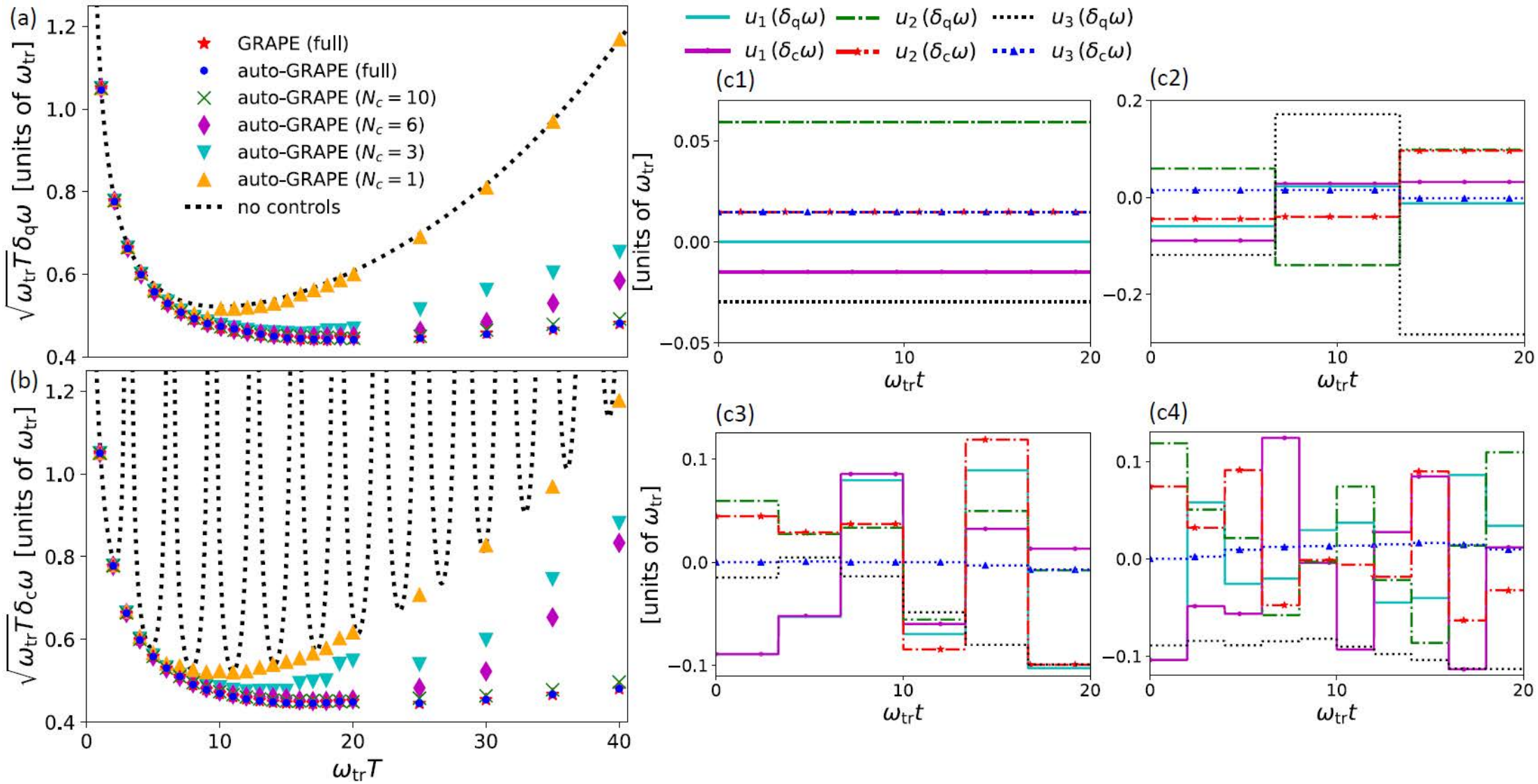
```
kwargs = {"Adam": True, "ctrl0":ctrl0, "max_episode": 300, \
          "epsilon": 0.01, "beta1": 0.90, "beta2": 0.99}
control = ControlOpt(savefile=False, method = "GRAPE", **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay, \
                 ctrl_bound=[-2.0, 2.0], dyn_method="expm")
control.QFIM()
```

auto-GRAPE

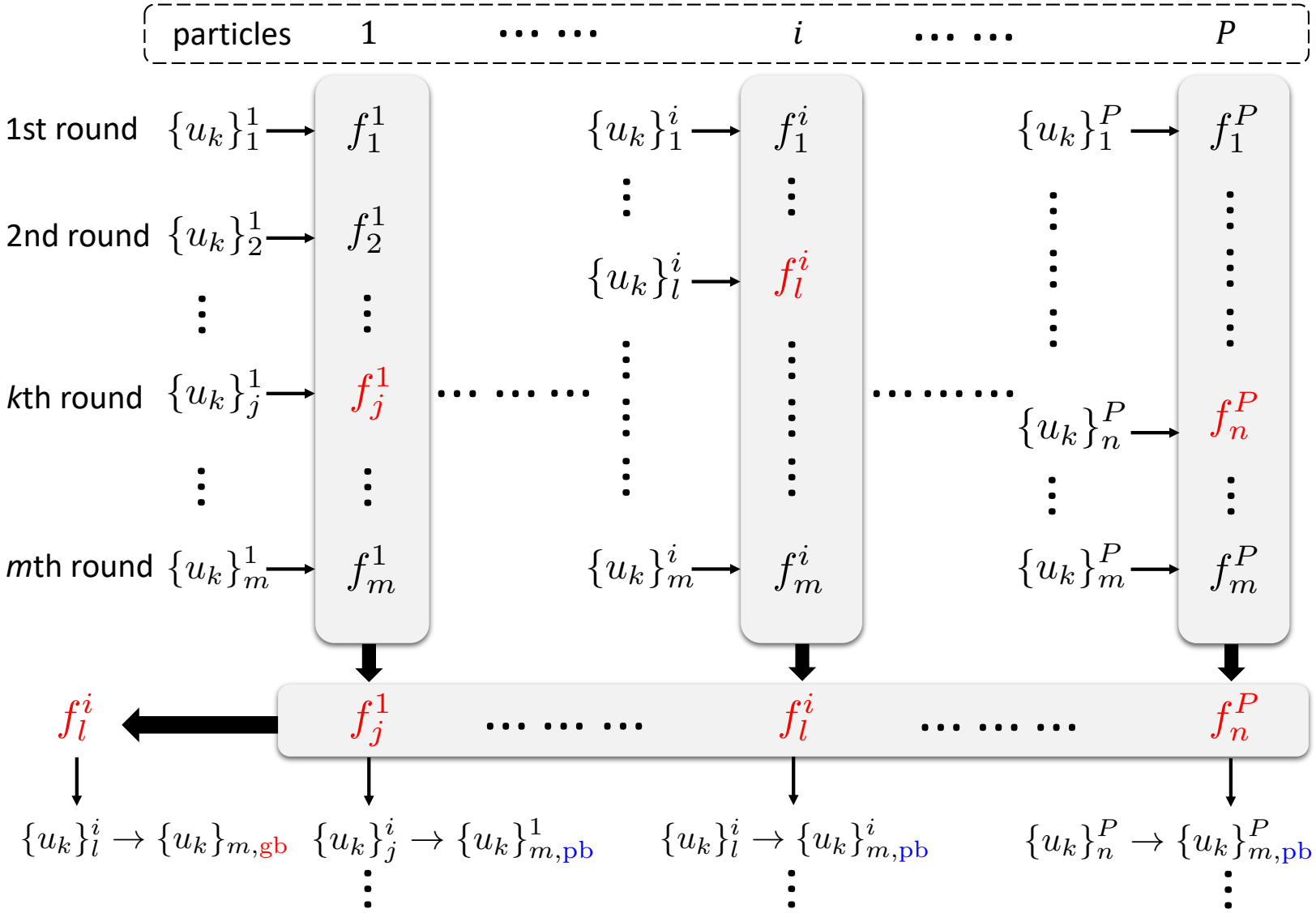
```
kwargs = {"Adam": True, "ctrl0":ctrl0, "max_episode": 300, \
          "epsilon": 0.01, "beta1": 0.90, "beta2": 0.99}
control = ControlOpt(savefile=False, method = "auto-GRAPE",\
                     **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay, \
                 ctrl_bound=[-2.0, 2.0], dyn_method="expm")
control.QFIM()
```

$\omega_0 T$	5	10	15	20	30	40
GRAPE	5.23 s	21.75 s	44.95 s	71.00 s	178.56 s	373.89 s
auto-GRAPE	0.93 s	1.36 s	2.13 s	3.37 s	5.21 s	5.96 s

Gradient ascent pulse engineering



Particle swarm optimization



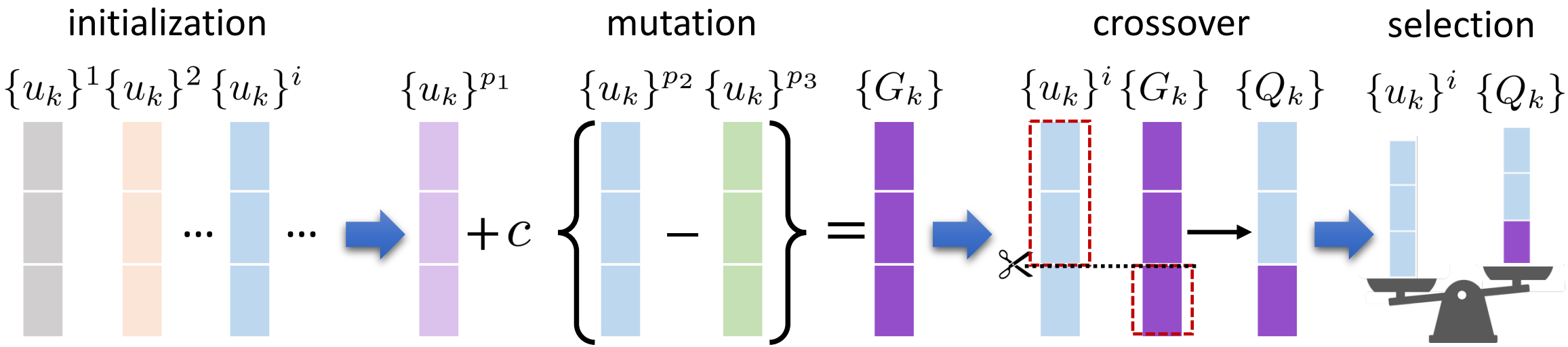
J. Kennedy and R. Eberhar, Proc. 1995 IEEE International Conference on Neural Networks **4**, 1942-1948 (1995).

R. C. Eberhart and Y. Shi, in Proc. 2001 Congr. Evol. Comput. (IEEE Cat. No.01TH8546) (IEEE, 2001), pp. 81–86.

Particle swarm optimization

```
kwargs = {"p_num":10, "ctrl0":ctrl0, "c0":1.0, "c1":2.0,\
          "c2":2.0, "max_episode":[1000,100], "seed":1234}
control = ControlOpt(savefile=False, method="PSO", **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay,\
                 ctrl_bound=[-2.0,2.0], dyn_method="expm")
control.QFIM()
```

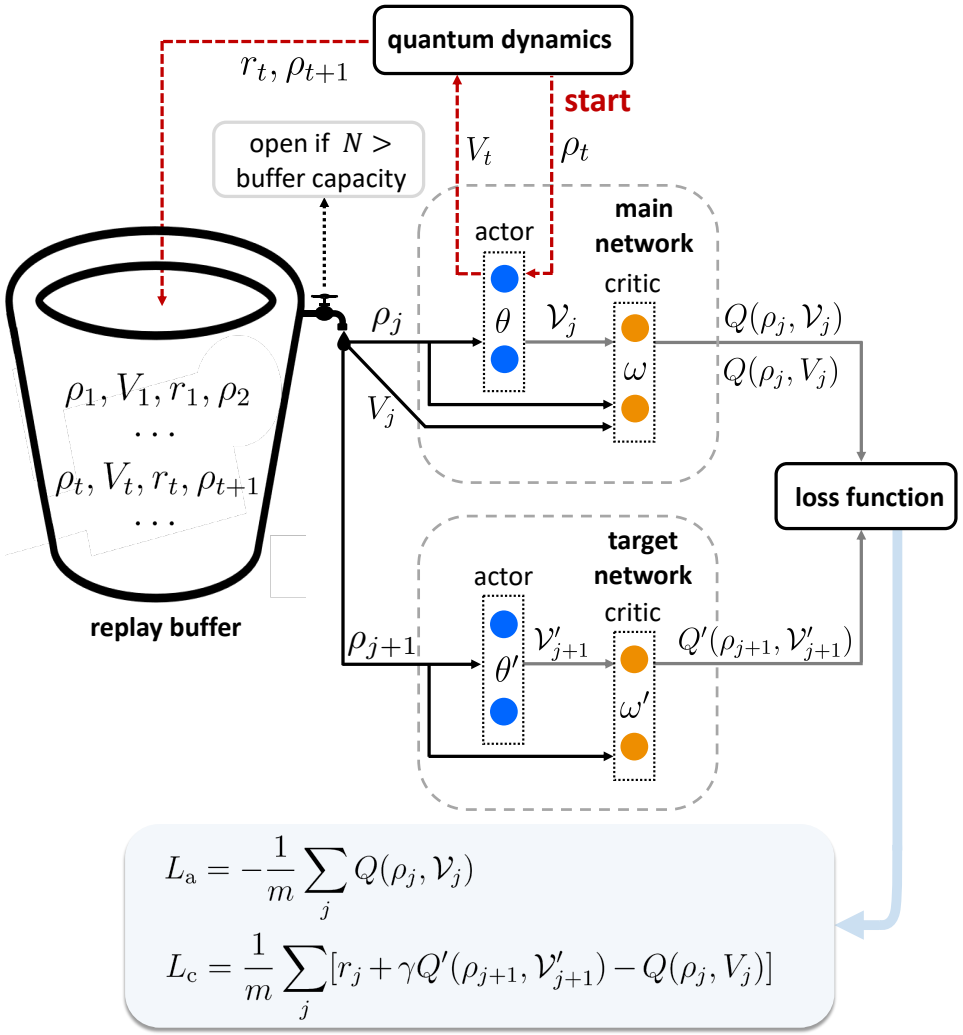
Differential evolution




```
kwargs = {"p_num":10, "ctrl0":ctrl0, "max_episode":1000,\
          "c":1.0, "cr":0.5, "seed":1234}
control = ControlOpt(savefile=False, method="DE", **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay,\
                 ctrl_bound=[-2.0,2.0], dyn_method="expm")
control.QFIM()
```

R. Storn and K. Price, Continuous Spaces, J. Global Optim. **11**, 341 (1997).

Deep deterministic policy gradients



$$L_{\mathrm{a}} = -\frac{1}{m}\sum_j Q(\rho_j, \mathcal{V}_j)$$

$$L_{\mathrm{c}} = \frac{1}{m}\sum_j [r_j + \gamma Q'(\rho_{j+1}, \mathcal{V}'_{j+1}) - Q(\rho_j, V_j)]$$

```
kwargs = {"layer_num":4, "layer_dim": 220, \
          "max_episode": 500, "seed": 1234}
control = ControlOpt(savefile=False, method="DDPG", **kwargs)
control.dynamics(tspan, rho0, H0, dH, Hc, decay=decay, \
                 ctrl_bound=[-2.0, 2.0], dyn_method="expm")
control.QFIM()
```

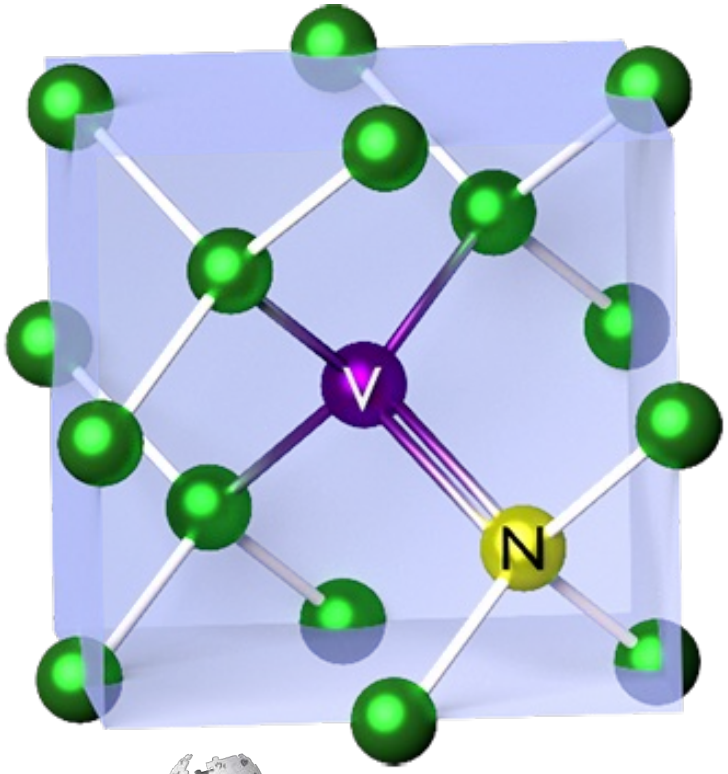

T. P. Lillicrap et al., arXiv:1509.02971.

Q.-S. Tan et al., Phys. Rev. A **103**, 032601 (2021).

J. Liu et al., Adv. Quantum Technol. **5**, 202100080 (2022).

Multiparameter estimation

Nitrogen-vacancy (NV) center in diamond



credit

It consists of a nearest-neighbor pair of a nitrogen atom, which substitutes for a carbon atom, and a lattice vacancy.

- NV centers have better spatial resolution
- NV centers have higher detection sensitivity
- NV centers can detect magnetic field, electric field, and etc.

Multiparameter estimation

magnetic interaction

hyperfine interaction

Hamiltonian $$H_0/\hbar = DS_3^2 + g_S \vec{B}\cdot\vec{S} + g_I \vec{B}\cdot\vec{I} + \vec{S}^T \mathcal{A}\vec{I}$$

zero-field term

nuclear Zeeman interaction

D is the zero-field-splitting parameter

$\hbar$ is the Plank's constant

$g_S = g_e \mu_B/\hbar$ g_e is the g factor of the nuclear

μ_B is the Bohr magneton

$g_I = g_n \mu_n/\hbar$ g_n is the g factor of the electron

μ_n is the nuclear magneton

$\vec{I} = (I_1, I_2, I_3)^T$ with $I_{1(2,3)} = \mathbb{1} \otimes \sigma_{1(2,3)}$ the nuclear operator

$\vec{S} = (S_1, S_2, S_3)^T$ with $S_{1(2,3)} = s_{1(2,3)} \otimes \mathbb{1}$ the electron operator

$\mathcal{A} = \mathrm{diag}(A_1, A_1, A_2)$ is the hyperfine tensor

Multiparameter estimation

Hamiltonian $$H/\hbar = DS_3^2 + g_S \vec{B}\cdot\vec{S} + g_I \vec{B}\cdot\vec{I} + \vec{S}^{\,T}\mathcal{A}\vec{I} + \sum_{i=1}^{3}\Omega_i(t)S_i$$

Dynamics $$\partial_t\rho = -i[H,\rho] + \frac{\gamma}{2}(S_3\rho S_3 - S_3^2\rho - \rho S_3^2)$$

Initial state $$\frac{1}{\sqrt{2}}(|0\rangle + |-1\rangle)\otimes|0\rangle$$

$$D = 2\pi \times 2.87\,\mathrm{GHz} \qquad A_1 = 2\pi \times 3.65\,\mathrm{MHz} \qquad B_1 = B_2 = B_3 = 0.50\,\mathrm{mT}$$
$$g_S = 2\pi \times 28.03\,\mathrm{GHz/T} \qquad A_2 = 2\pi \times 3.03\,\mathrm{MHz} \qquad \Omega_i(t) \in [-20\,\mathrm{MHz}, 20\,\mathrm{MHz}]$$
$$g_I = 2\pi \times 4.32\,\mathrm{MHz/T} \qquad \gamma = 2\pi \times 1\,\mathrm{MHz}$$

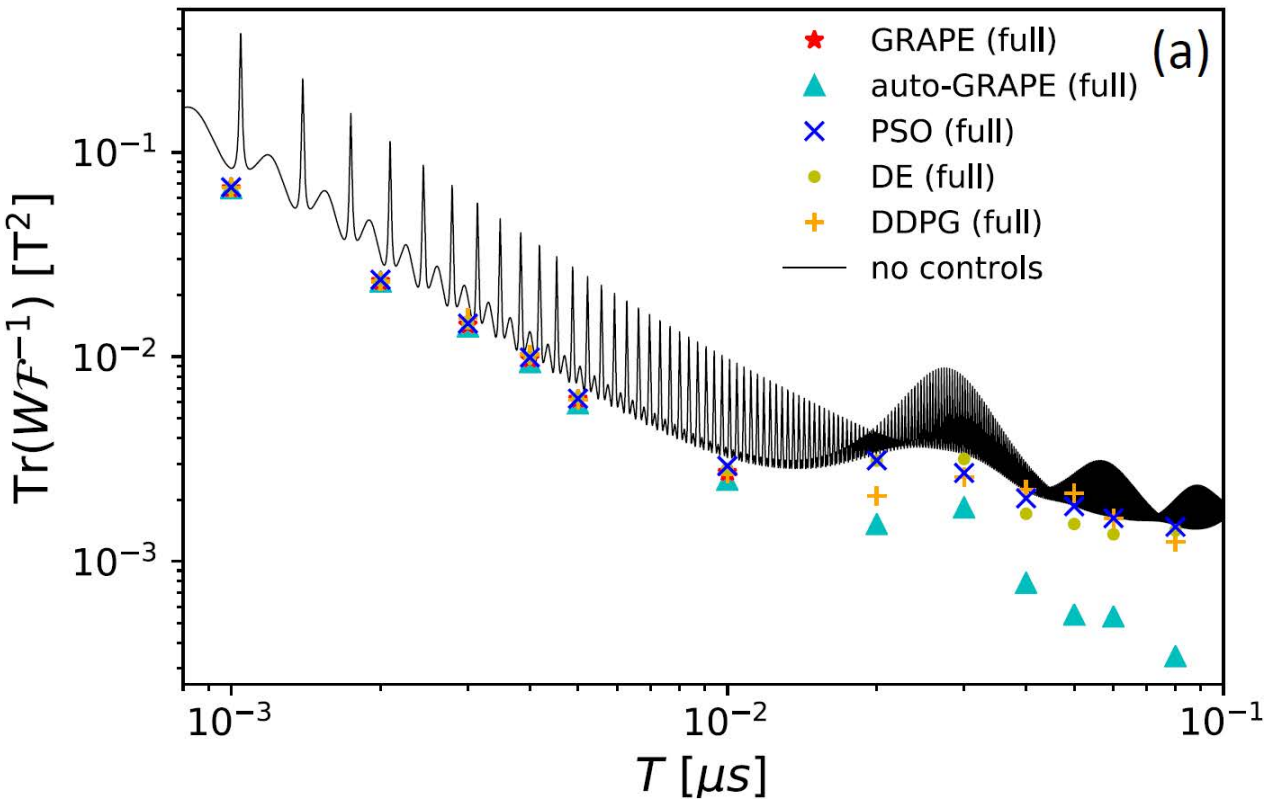
J. F. Barry et al., Rev. Mod. Phys. **92**, 015004 (2020).

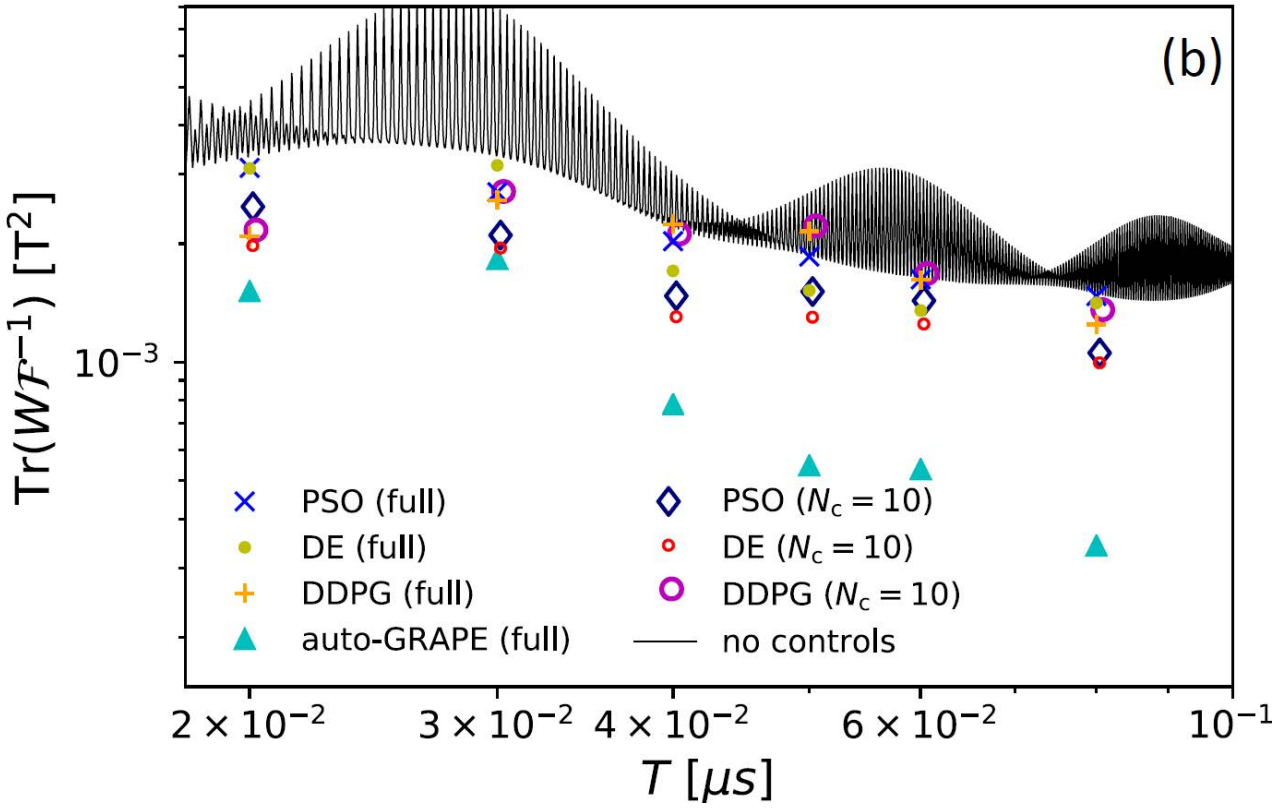
Multiparameter estimation

```
[ ]: from quanestimation import *
import numpy as np
from numpy.random import random
# free Hamiltonian
sx = np.array([[0., 1.],[1., 0.]])
sy = np.array([[0., -1.j],[1.j, 0.]])
sz = np.array([[1., 0.],[0., -1.]])
ide2 = np.identity(2)
s1 = np.array([[0., 1., 0.],\
               [1., 0., 1.],\
               [0., 1., 0.]])/np.sqrt(2)
s2 = np.array([[0., -1.j, 0.],\
               [1.j, 0., -1.j],\
               [0., 1.j, 0.]])/np.sqrt(2)
s3 = np.array([[1., 0., 0.],\
               [0., 0., 0.],\
               [0., 0., -1.]])
ide3 = np.identity(3)
I1, I2, I3 = np.kron(ide3, sx), np.kron(ide3, sy), np.kron(ide3, sz)
S1, S2, S3 = np.kron(s1, ide2), np.kron(s2, ide2), np.kron(s3, ide2)
B1, B2, B3 = 5.0e-4, 5.0e-4, 5.0e-4
cons = 100
D = (2*np.pi*2.87*1000)/cons
gS, gI = (2*np.pi*28.03*1000)/cons, (2*np.pi*4.32)/cons
A1, A2 = (2*np.pi*3.65)/cons, (2*np.pi*3.03)/cons
H0 = D*np.kron(np.dot(s3, s3), ide2) + gS*(B1*S1+B2*S2+B3*S3) \
     + gI*(B1*I1+B2*I2+B3*I3) + A1*(np.kron(s1, sx) + np.kron(s2, sy))
     + A2*np.kron(s3, sz)
dH = [gS*S1+gI*I1, gS*S2+gI*I2, gS*S3+gI*I3]
```

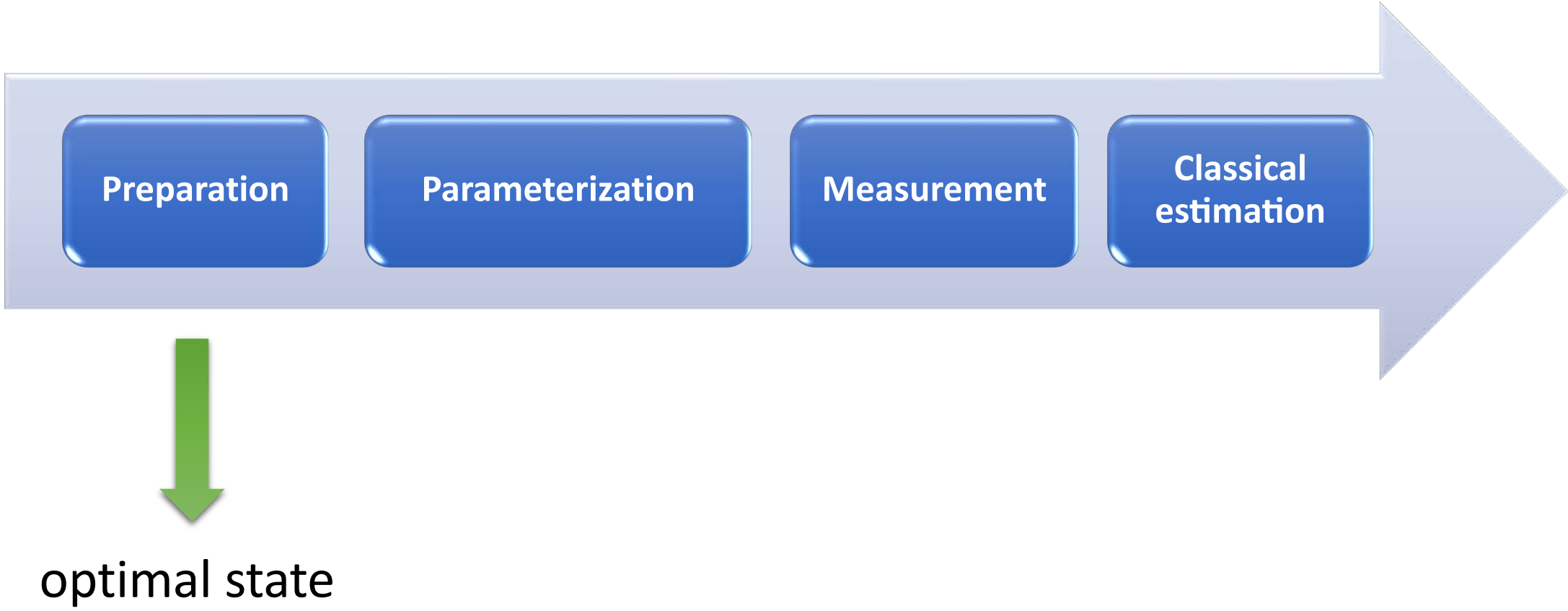
```
[ ]: # initial state
rho0 = np.zeros((6, 6), dtype=np.complex128)
rho0[0][0], rho0[0][4],  = 0.5, 0.5
rho0[4][0], rho0[4][4] = 0.5, 0.5
# decay
decay = [[S3, 2*np.pi/cons]]
# time span for the evolution
tspan = np.linspace(0., 2., 4000)
# measurement
dim = len(rho0)
M = [
    np.dot(basis(dim, i), basis(dim, i).conj().T)
    for i in range(dim)
]
# guessed control coefficients
cnum = 10
np.random.seed(1234)
ini_1 = np.zeros((len(Hc),cnum))
ini_2 = -0.2*np.random.random ((len(Hc), cnum))
ctrl0 = [ini_1, ini_2]
# control Hamiltonians
Hc = [S1, S2, S3]
```

Multiparameter estimation





State optimization



State optimization

Pure state optimization $|\psi\rangle = \sum_i c_i |i\rangle$

Search a set of normalized complex coefficients $\{c_i\}$

```
state = StateOpt(savefile=False,method="AD",**kwargs)
state.dynamics(tspan,H0,dH,Hc=[],ctrl=[],decay=[]
               dyn_method="expm")
state.QFIM(W=[],LDtype="SLD")
state.CFIM(M=[],W=[])
state.HCRB(W=[])
```

State optimization

State optimization algorithms in QuanEstimation

Algorithms	method=	**kwargs and default values	
AD	"AD"	"Adam"	False
		"psi0"	[]
		"max_episode"	300
		"epsilon"	0.01
		"beta1"	0.90
		"beta2"	0.99
PSO	"PSO"	"p_num"	10
		"psi0"	[]
		"max_episode"	[1000,100]
		"c0"	1.0
		"c1"	2.0
		"c2"	2.0
		"seed"	1234
DE	"DE"	"p_num"	10
		"psi0"	[]
		"max_episode"	1000
		"c"	1.0
		"cr"	0.5
		"seed"	1234

Algorithms	method=	**kwargs and default values	
NM	"NM"	"p_num"	10
		"psi0"	[]
		"max_episode"	1000
		"ar"	1.0
		"ae"	2.0
		"ac"	0.5
		"as0"	0.5
		"seed"	1234
RI	"RI"	"psi0"	[]
		"max_episode"	300
		"seed"	1234
DDPG	"DDPG"	"psi0"	[]
		"max_episode"	500
		"layer_num"	3
		"layer_dim"	200
		"seed"	1234

State optimization

Lipkin-Meshkov-Glick model

$$H_{\mathrm{LMG}} = -\frac{\lambda}{N}(J_1^2 + gJ_2^2) - hJ_3$$

external transverse magnetic field

a set of *N* spin-1/2 mutually interaction through a *XY*-like Hamiltonian

The LMG model can be used to study

- quantum phase transitions,
- entanglement,
- …

H. J. Lipkin, N. Meshkov, and A. J. Glick, Nucl. Phys. **62**, 188 (1965).

State optimization

Hamiltonian

$$H_{\mathrm{LMG}} = -\frac{\lambda}{N}(J_1^2 + gJ_2^2) - hJ_3$$

Dynamics

$$\partial_t \rho = -i[H_{\mathrm{LMG}}, \rho] + \gamma\left(J_3 \rho J_3 - \frac{1}{2}\left\{\rho, J_3^2\right\}\right)$$

Initial state

$$|\theta, \phi\rangle = \exp\left(-\frac{\theta}{2}e^{-i\phi}J_+ + \frac{\theta}{2}e^{i\phi}J_-\right)|J, J\rangle$$

Pure state optimization

$$|\psi\rangle = \sum_{m=-J}^{J} c_m |J, m\rangle$$

$J_i = \sum_{j=1}^{N} \sigma_i^{(j)}/2 \ \ (i = 1, 2, 3) \quad J_\pm = J_1 \pm iJ_2$

$|J, J\rangle$ and $|J, m\rangle$ are the Dicke states

```
[ ]: from quanestimation import *
     import numpy as np
     from qutip import *

     # the dimension of the system
     N = 8
     # generation of the coherent spin state
     psi_css = spin_coherent(0.5*N, 0.5*np.pi, 0.5*np.pi,
                             type="ket").full()
     psi_css = psi_css.reshape(1, -1)[0]
     # guessed state
     psi0 = [psi_css]

     # Hamiltonian
     Lambda, g, h = 1.0, 0.5, 0.1
     Jx, Jy, Jz = jmat(0.5*N)
     Jx, Jy, Jz = Jx.full(), Jy.full(), Jz.full()
     H0 = -Lambda*(np.dot(Jx, Jx) + g*np.dot(Jy, Jy))/N - h*Jz
     dH = [-Lambda*np.dot(Jy, Jy)/N, -Jz]

     # decay
     decay = [[Jz, 0.1]]
     # time span for the evolution
     tspan = np.linspace(0., 10., 2500)
```

State optimization: noiseless scenario

AD algorithm

```
kwargs = {"Adam":False, "psi0":[psi_css], "epsilon":0.01,\
          "max_episode":300, "beta1":0.90, "beta2":0.99}
state = StateOpt(savefile=False, method="AD", **kwargs)
state.dynamics(tspan, H0, [dH[0]], decay=[],\
               dyn_method="expm")
state.QFIM()
```

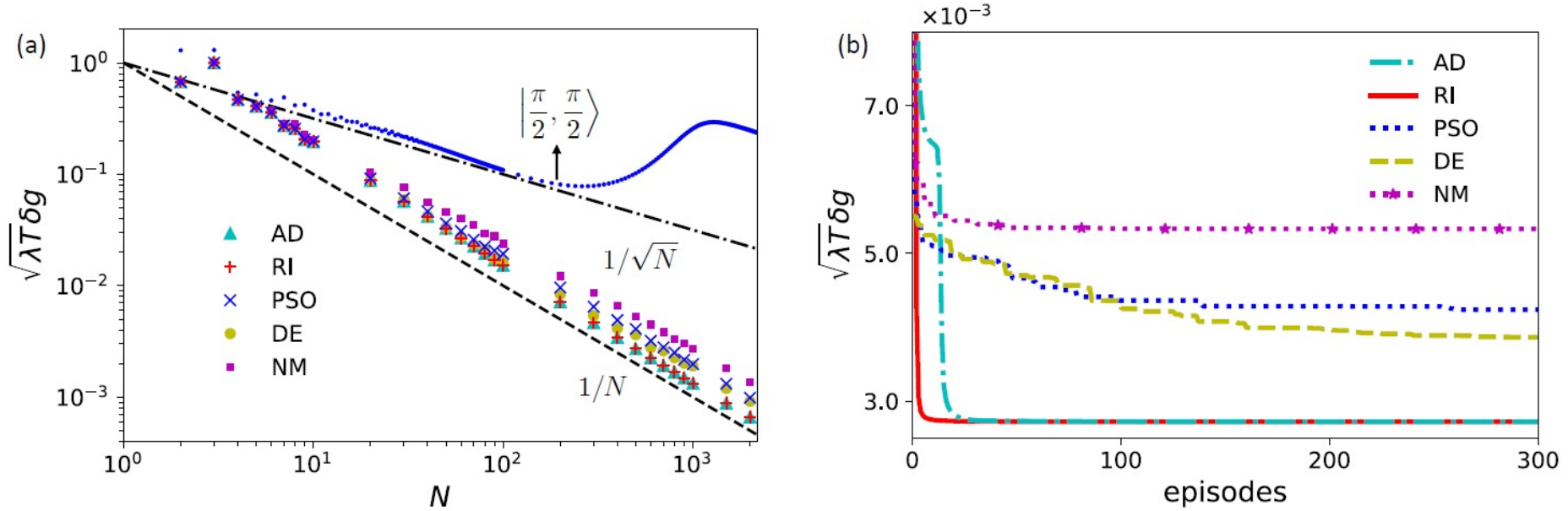
PSO algorithm

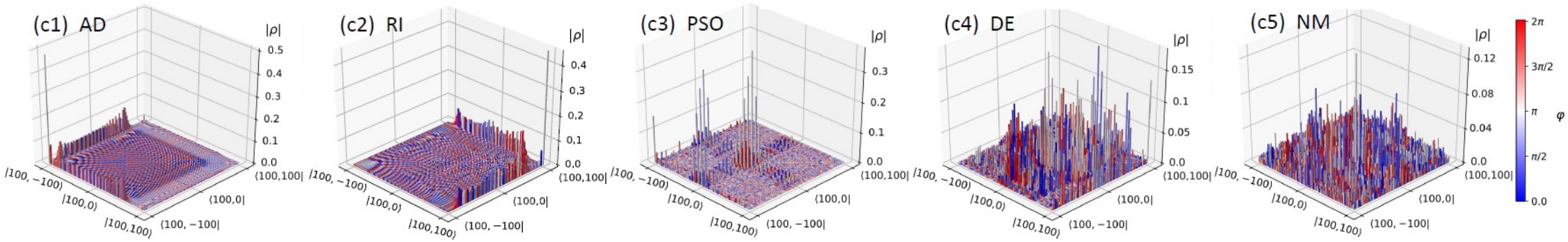
```
kwargs = {"p_num":10, "psi0":[psi_css], "c0":1.0, "c1":2.0,\
          "c2":2.0, "max_episode":[1000,100], "seed"=1234}
state = StateOpt(savefile=False, method="PSO", **kwargs)
state.dynamics(tspan, H0, [dH[0]], decay=[],\
               dyn_method="expm")
state.QFIM()
```

...

State optimization: noiseless scenario

$$H_{\mathrm{LMG}} = -\frac{\lambda}{N}(J_1^2 + gJ_2^2) - hJ_3$$





State optimization: noisy scenario

AD algorithm

```
kwargs = {"Adam":False, "psi0":[psi_css], "epsilon":0.01,\
          "max_episode":300, "beta1":0.90, "beta2":0.99}
state = StateOpt(savefile=False, method="AD", **kwargs)
state.dynamics(tspan, H0, [dH[0]], decay=decay,\
               dyn_method="expm")
state.QFIM()
```

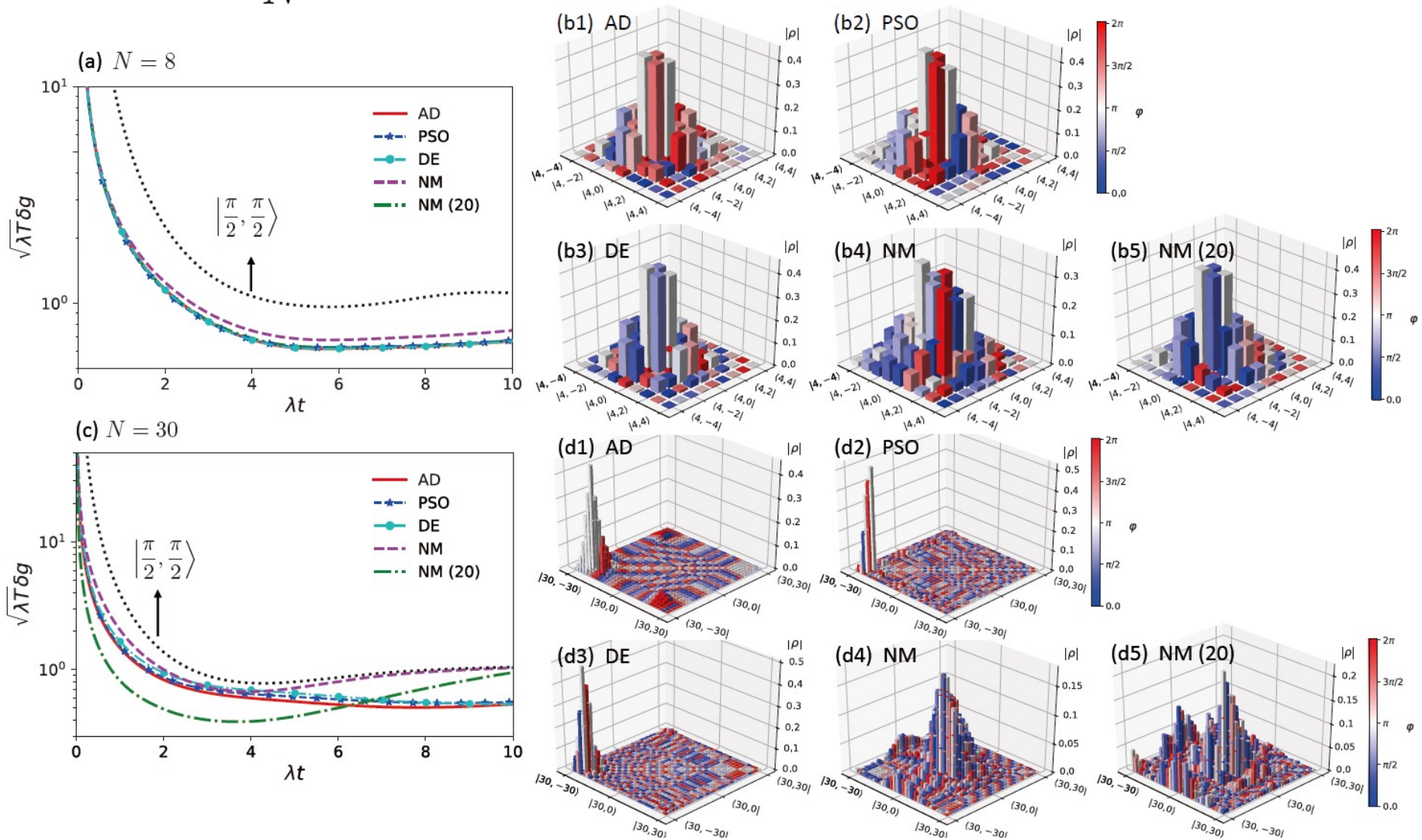
DE algorithm

```
kwargs = {"p_num":10, "psi0":[psi_css], "max_episode":1000,\
          "c":1.0, "cr":0.5, "seed"=1234}
state = StateOpt(savefile=False, method="DE", **kwargs)
state.dynamics(tspan, H0, [dH[0]], decay=decay,\
               dyn_method="expm")
state.QFIM()
```

...

State optimization: noisy scenario

$$H_{\mathrm{LMG}} = -\frac{\lambda}{N}(J_1^2 + gJ_2^2) - hJ_3$$



State optimization: multiparameter scenario

AD algorithm

```
kwargs = {"Adam":False, "psi0":[psi_css], "epsilon":0.01,\
          "max_episode":300, "beta1":0.90, "beta2":0.99}
state = StateOpt(savefile=False, method="AD", **kwargs)
state.dynamics(tspan, H0, dH, decay=decay,dyn_method="expm")
state.QFIM(W=W)
```

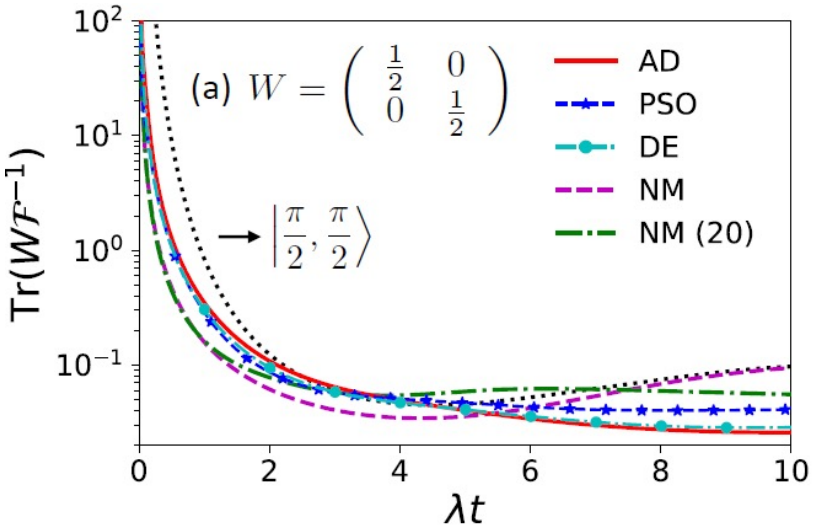
NM algorithm

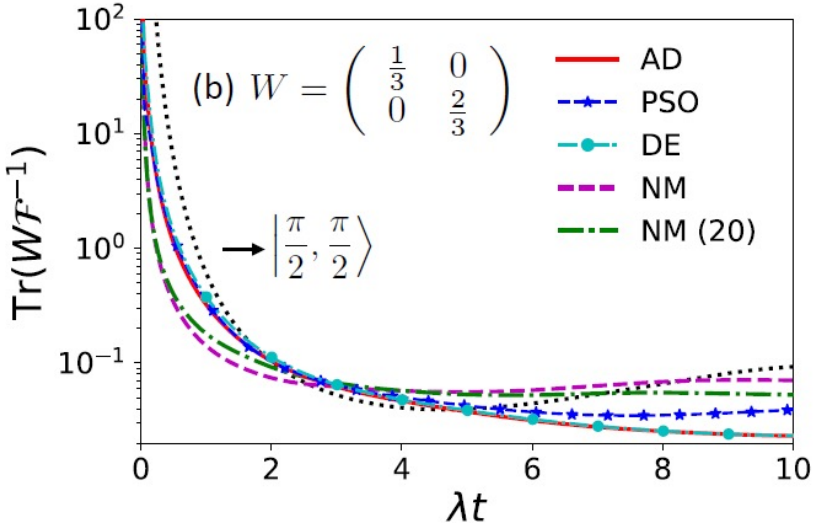
```
kwargs = {"p_num":200, "psi0":[psi_css], "max_episode"
   :1000,\
"ar":1.0, "ae":2.0, "ac"=0.5, "as0":0.5,\
"seed":1234}
state = StateOpt(savefile=False, method="NM", **kwargs)
state.dynamics(tspan, H0, dH, decay=decay,dyn_method="expm")
state.QFIM(W=W)
```

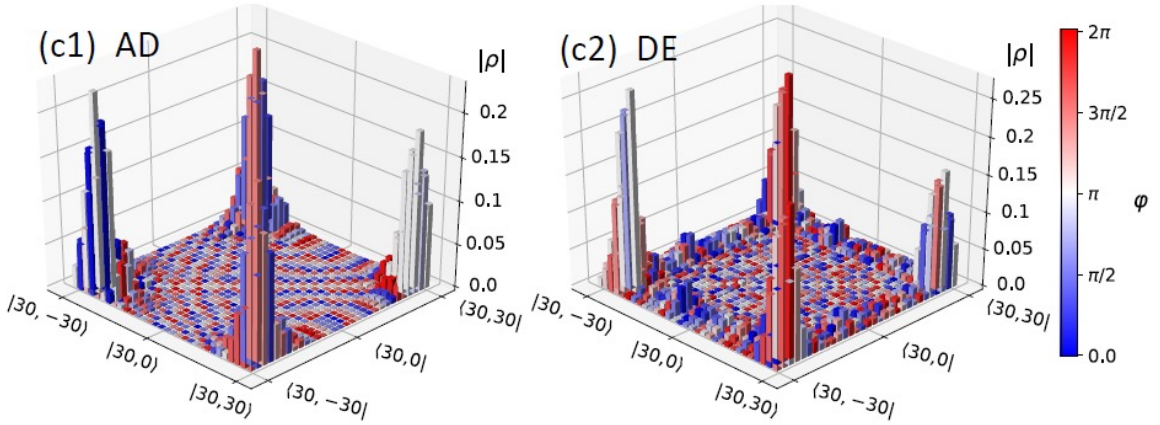
...

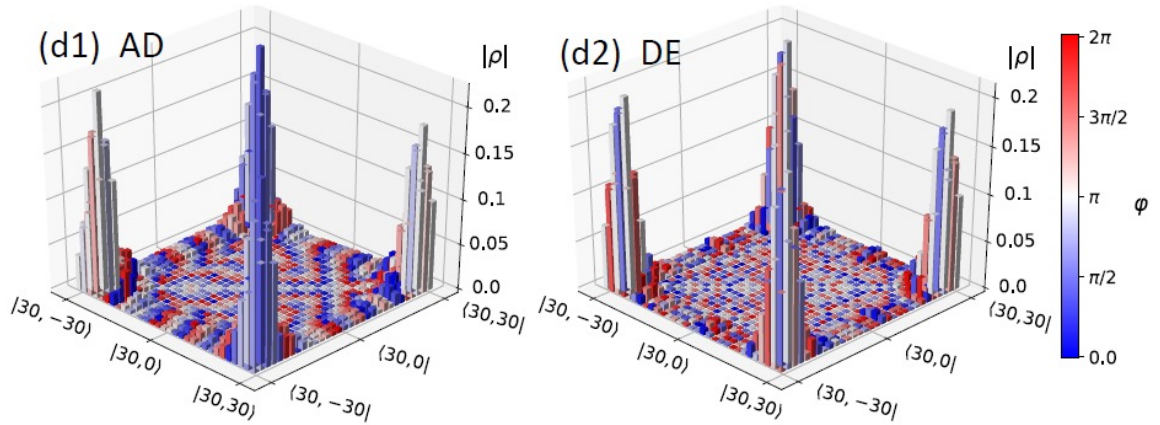
State optimization: multiparameter scenario

$$H_{\mathrm{LMG}} = -\frac{\lambda}{N}(J_1^2 + gJ_2^2) - hJ_3$$

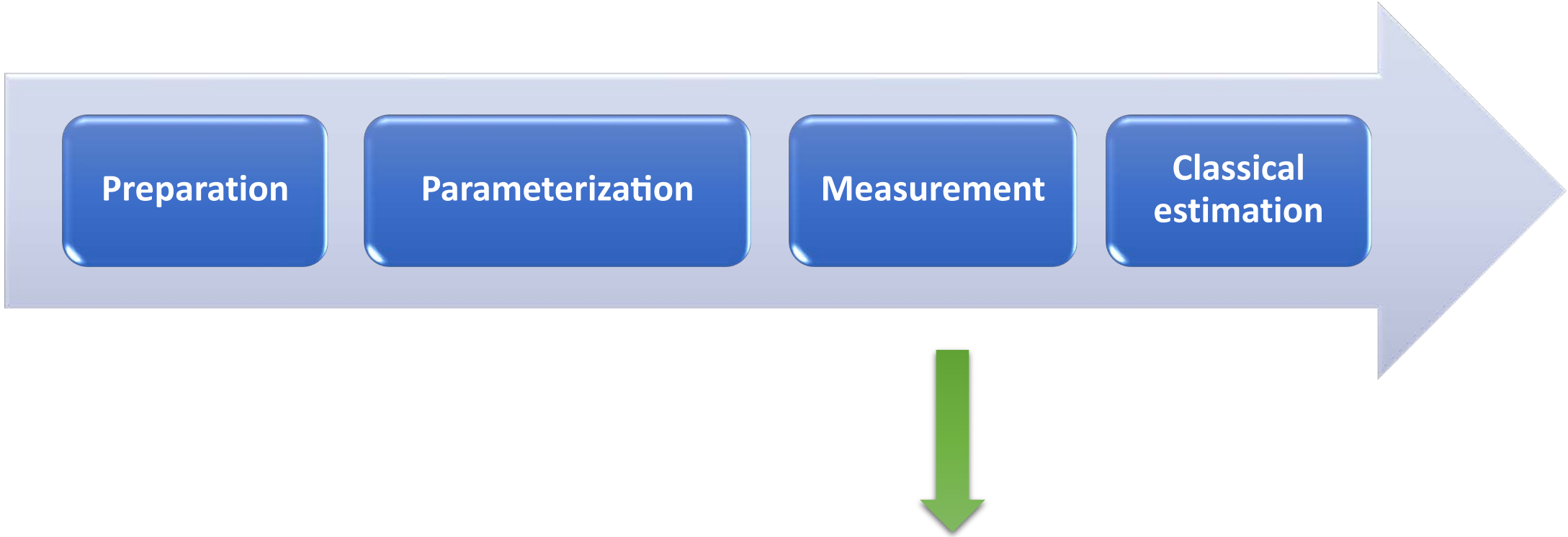








Measurement optimization



Measurement optimization

Projection

$$\{\Pi_i\}_{i=1}^n \quad \Pi_i = |\phi_i\rangle\langle\phi_i| \quad |\phi_i\rangle = \sum_j C_{ij}|j\rangle$$

Linear combination

$$\{\Pi_i\}_{i=1}^n \longrightarrow \{\Pi_i'\}_{i=1}^m \quad \Pi_i' = \sum_{j=1}^n B_{ij}\Pi_j$$

Rotation

$$\{\Pi_i\}_{i=1}^n \longrightarrow \{U\Pi_i U^\dagger\}_{i=1}^n \quad U = \prod_k \exp(is_k\lambda_k)$$

```
m = MeasurementOpt(mtype="projection",minput=[],\
                   savefile=False,method="DE",**kwargs)
m.dynamics(tspan,rho0,H0,dH,Hc=[],ctrl=[],decay=[])
m.CFIM(W=[])
```

Measurement optimization algorithms

Algorithms	method=	**kwargs and default values	
PSO	"PSO"	"p_num"	10
		"measurement0"	[]
		"max_episode"	[1000,100]
		"c0"	1.0
		"c1"	2.0
		"c2"	2.0
		"seed"	1234
DE	"DE"	"p_num"	10
		"measurement0"	[]
		"max_episode"	1000
		"c"	1.0
		"cr"	0.5
		"seed"	1234
AD (available when *mtype* = *"input"*)	"AD"	"Adam"	False
		"measurement0"	[]
		"max_episode"	300
		"epsilon"	0.01
		"beta1"	0.90
		"beta2"	0.99

Measurement optimization

Hamiltonian $H = \frac{1}{2}\omega_0\sigma_3$

Dynamics

$$\partial_t\rho = -i\left[H,\rho\right] + \gamma_+\left(\sigma_+\rho\sigma_- - \frac{1}{2}\left\{\sigma_-\sigma_+,\rho\right\}\right) + \gamma_-\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\left\{\sigma_+\sigma_-,\rho\right\}\right)$$

Initial state $|+\rangle$

```
[ ]: import numpy as np
     from quanestimation import *
     from qutip import *

     # initial state
     rho0 = 0.5 * np.ones((2, 2))
     # Hamiltonian
     omega = 1.0
     s1, s2, s3 = sigmax(), sigmay(), sigmaz()
     s1, s2, s3 = s1.full(), s2.full(), s3.full()
     H0 = 0.5* omega * s3
     dH = [0.5 * s3]
     # decay
     sp = destroy(2).full()
     sm = create(2).full()
     decay = [[sp, 0.], [sm, 0.1]]

     # time for evolution
     tspan = np. linspace (0., 10., 2500)
```

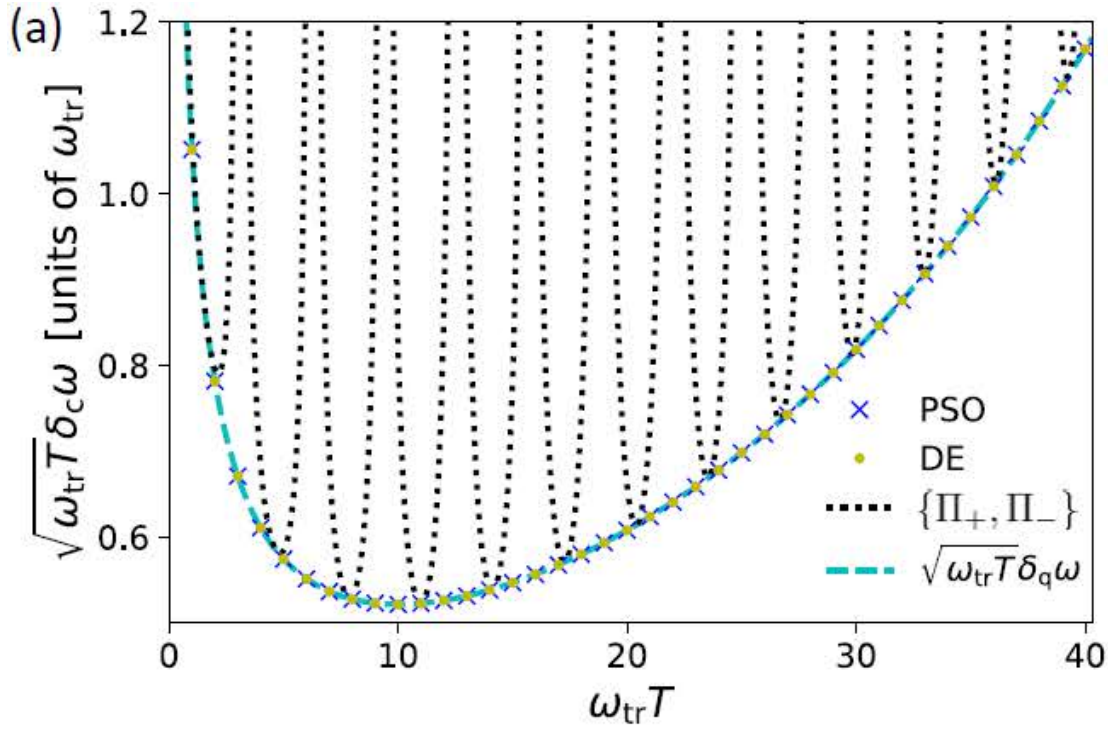
Measurement optimization

DE algorithm

```
kwargs = {"p_num":10, "measurement0":[], "seed":1234,\
          "max_episode":1000,"c":1.0, "cr":0.5}
m = MeasurementOpt(mtype="projection",minput=[],\
                savefile=False, method="DE", **kwargs)
m.dynamics(tspan, rho0, H0, dH, decay=decay,\
           dyn_method="expm")
m.CFIM()
```

PSO algorithm

```
kwargs = {"p_num":10, "measurement0":[], "c0"=1.0,\
          "c1":2.0,"c2":2.0, "max_episode":[1000,100],\
          "seed"=1234}
m = MeasurementOpt(mtype="projection",minput=[],\
                savefile=False, method="PSO", **kwargs)
m.dynamics(tspan, rho0, H0, dH, decay=decay,\
           dyn_method="expm")
m.CFIM()
```



Measurement optimization

```
from quanestimation import *
import numpy as np
from numpy.random import random
# free Hamiltonian
sx = np.array([[0., 1.],[1., 0.]])
sy = np.array([[0., -1.j],[1.j, 0.]])
sz = np.array([[1., 0.],[0., -1.]])
ide2 = np.identity(2)
s1 = np.array([[0., 1., 0.],\
               [1., 0., 1.],\
               [0., 1., 0.]])/np.sqrt(2)
s2 = np.array([[0., -1.j, 0.],\
               [1.j, 0., -1.j],\
               [0., 1.j, 0.]])/np.sqrt(2)
s3 = np.array([[1., 0., 0.],\
               [0., 0., 0.],\
               [0., 0., -1.]])
ide3 = np.identity(3)
I1, I2, I3 = np.kron(ide3, sx), np.kron(ide3, sy), np.kron(ide3, sz)
S1, S2, S3 = np.kron(s1, ide2), np.kron(s2, ide2), np.kron(s3, ide2)
B1, B2, B3 = 5.0e-4, 5.0e-4, 5.0e-4
cons = 100
D = (2*np.pi*2.87*1000)/cons
gS, gI = (2*np.pi*28.03*1000)/cons, (2*np.pi*4.32)/cons
A1, A2 = (2*np.pi*3.65)/cons, (2*np.pi*3.03)/cons
H0 = D*np.kron(np.dot(s3, s3), ide2) + gS*(B1*S1+B2*S2+B3*S3) \
     + gI*(B1*I1+B2*I2+B3*I3) + A1*(np.kron(s1, sx) + np.kron(s2, sy))
     + A2*np.kron(s3, sz)
dH = [gS*S1+gI*I1, gS*S2+gI*I2, gS*S3+gI*I3]
```

Hamiltonian

$$H/\hbar = DS_3^2 + g_{\mathrm{S}}\vec{B}\cdot\vec{S} + g_{\mathrm{I}}\vec{B}\cdot\vec{I} + \vec{S}^{\mathrm{T}}\mathcal{A}\vec{I}$$

Dynamics

$$\partial_t\rho = -i[H,\rho] + \frac{\gamma}{2}(S_3\rho S_3 - S_3^2\rho - \rho S_3^2)$$

Initial state

$$\frac{1}{\sqrt{2}}(|0\rangle + |-1\rangle)\otimes|0\rangle$$

[ ]:

```
# initial state
rho0 = np.zeros((6, 6), dtype=np.complex128)
rho0[0][0], rho0[0][4]  = 0.5, 0.5
rho0[4][0], rho0[4][4] = 0.5, 0.5

# decay
decay = [[S3, 2*np.pi/cons]]

# time span for the evolution
tspan = np.linspace(0., 2., 4000)
```

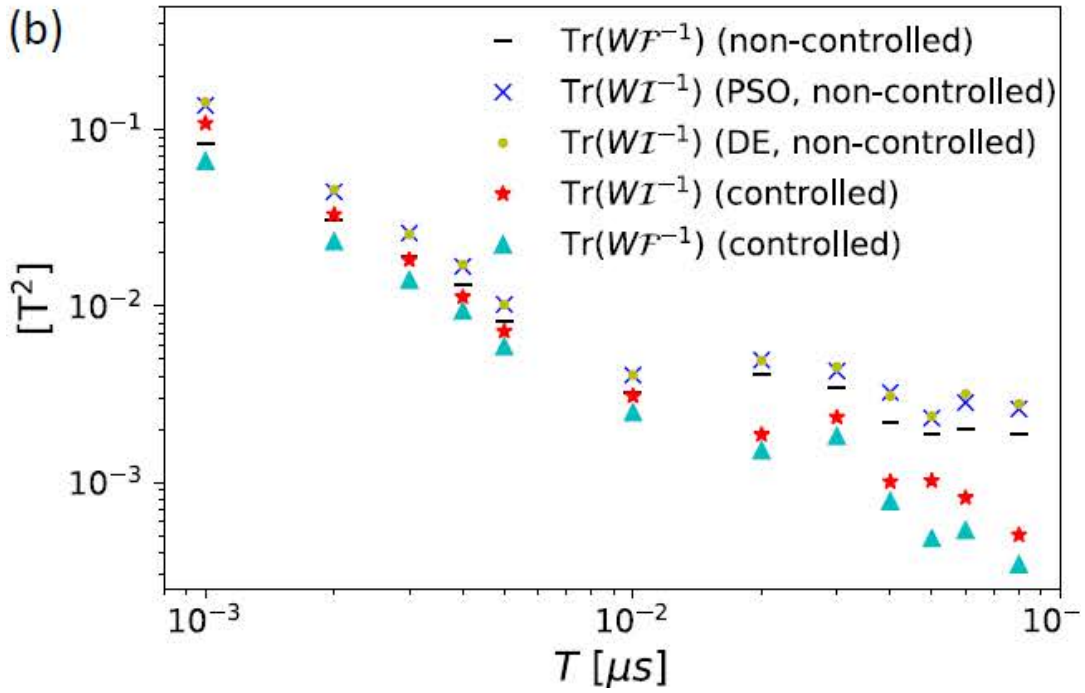
Measurement optimization

DE algorithm

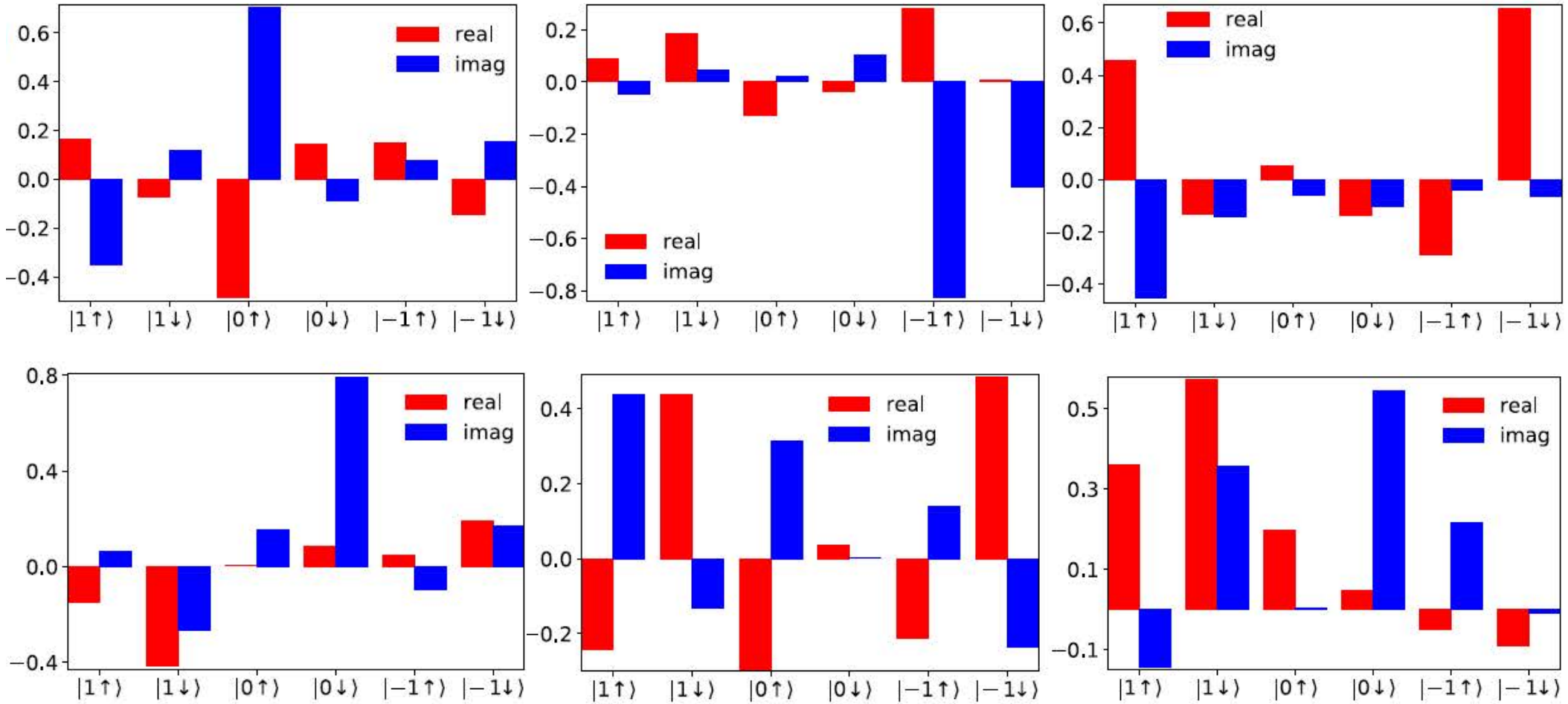
```
kwargs = {"p_num":10, "measurement0":[], "seed":1234,\
          "max_episode":1000,"c":1.0, "cr":0.5}
m = MeasurementOpt(mtype="projection",minput=[],\
                savefile=False, method="DE", **kwargs)
m.dynamics(tspan, rho0, H0, dH, decay=decay,\
           dyn_method="expm")
m.CFIM()
```

PSO algorithm

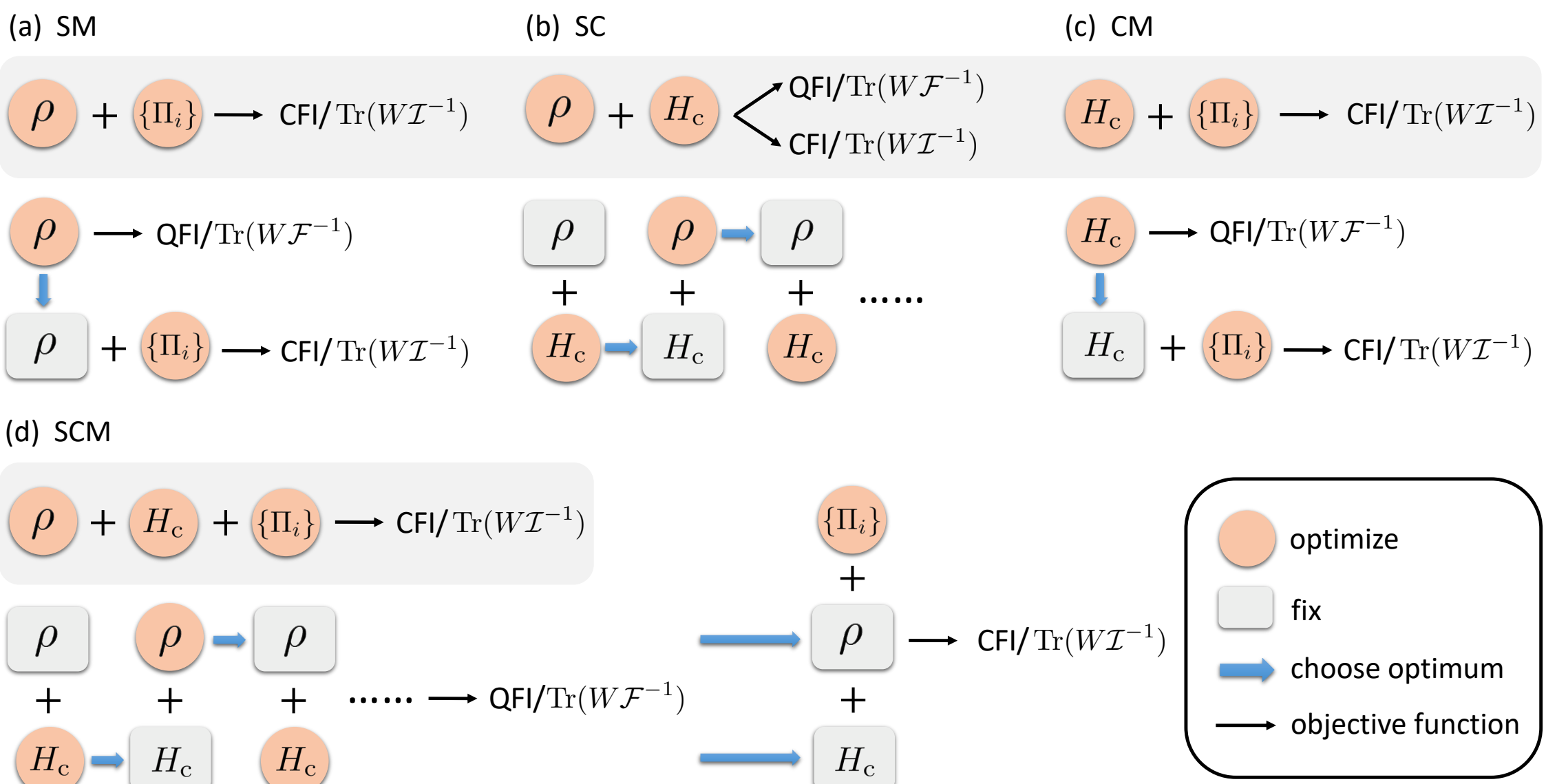
```
kwargs = {"p_num":10, "measurement0":[], "c0"=1.0,\
          "c1":2.0,"c2":2.0, "max_episode":[1000,100],\
          "seed"=1234}
m = MeasurementOpt(mtype="projection",minput=[],\
                savefile=False, method="PSO", **kwargs)
m.dynamics(tspan, rho0, H0, dH, decay=decay,\
           dyn_method="expm")
m.CFIM()
```



Measurement optimization



Comprehensive optimization



Comprehensive optimization

Comprehensive optimization algorithms

Algorithms	method=	**kwargs and default values	
PSO	"PSO"	"p_num"	10
		"psi0"	[]
		"ctrl0"	[]
		"measurement0"	[]
		"max_episode"	[1000,100]
		"c0"	1.0
		"c1"	2.0
		"c2"	2.0
		"seed"	1234
DE	"DE"	"p_num"	10
		"psi0"	[]
		"ctrl0"	[]
		"measurement0"	[]
		"max_episode"	1000
		"c"	1.0
		"cr"	0.5
		"seed"	1234
AD (available for SC)	"AD"	"Adam"	False
		"psi0"	[]
		"ctrl0"	[]
		"measurement0"	[]
		"max_episode"	300
		"epsilon"	0.01
		"beta1"	0.90
		"beta2"	0.99

```
com = ComprehensiveOpt(savefile=False,method="DE",**kwargs)
com.dynamics(tspan,H0,dH,Hc=[],ctrl=[],decay=[],\
             ctrl_bound=[], dyn_method="expm")
com.SE(W=[])
com.SC(W=[],M=[],target="QFIM",LDtype="SLD")
com.CM(rho0,W=[])
com.SCM(W=[])
```

Comprehensive optimization

```
from quanestimation import *
import numpy as np
from numpy.random import random
# free Hamiltonian
sx = np.array([[0., 1.],[1., 0.]])
sy = np.array([[0., -1.j],[1.j, 0.]])
sz = np.array([[1., 0.],[0., -1.]])
ide2 = np.identity(2)
s1 = np.array([[0., 1., 0.],\
               [1., 0., 1.],\
               [0., 1., 0.]])/np.sqrt(2)
s2 = np.array([[0., -1.j, 0.],\
               [1.j, 0., -1.j],\
               [0., 1.j, 0.]])/np.sqrt(2)
s3 = np.array([[1., 0., 0.],\
               [0., 0., 0.],\
               [0., 0., -1.]])
ide3 = np.identity(3)
I1, I2, I3 = np.kron(ide3, sx), np.kron(ide3, sy), np.kron(ide3, sz)
S1, S2, S3 = np.kron(s1, ide2), np.kron(s2, ide2), np.kron(s3, ide2)
B1, B2, B3 = 5.0e-4, 5.0e-4, 5.0e-4
cons = 100
D = (2*np.pi*2.87*1000)/cons
gS, gI = (2*np.pi*28.03*1000)/cons, (2*np.pi*4.32)/cons
A1, A2 = (2*np.pi*3.65)/cons, (2*np.pi*3.03)/cons
H0 = D*np.kron(np.dot(s3, s3), ide2) + gS*(B1*S1+B2*S2+B3*S3) \
     + gI*(B1*I1+B2*I2+B3*I3) + A1*(np.kron(s1, sx) + np.kron(s2, sy))
     + A2*np.kron(s3, sz)
dH = [gS*S1+gI*I1, gS*S2+gI*I2, gS*S3+gI*I3]
```

Hamiltonian

$$H/\hbar = DS_3^2 + g_{\mathrm{S}}\vec{B}\cdot\vec{S} + g_{\mathrm{I}}\vec{B}\cdot\vec{I} + \vec{S}^{\mathrm{T}}\mathcal{A}\vec{I} + \sum_{i=1}^{3}\Omega_i(t)S_i$$

Dynamics

$$\partial_t\rho = -i[H,\rho] + \frac{\gamma}{2}(S_3\rho S_3 - S_3^2\rho - \rho S_3^2)$$

Initial state

$$\frac{1}{\sqrt{2}}(|0\rangle + |-1\rangle)\otimes|0\rangle$$

[ ]:

```
# initial state
rho0 = np.zeros((6, 6), dtype=np.complex128)
rho0[0][0], rho0[0][4]  = 0.5, 0.5
rho0[4][0], rho0[4][4] = 0.5, 0.5

# decay
decay = [[S3, 2*np.pi/cons]]

# time span for the evolution
tspan = np.linspace(0., 2., 4000)
```

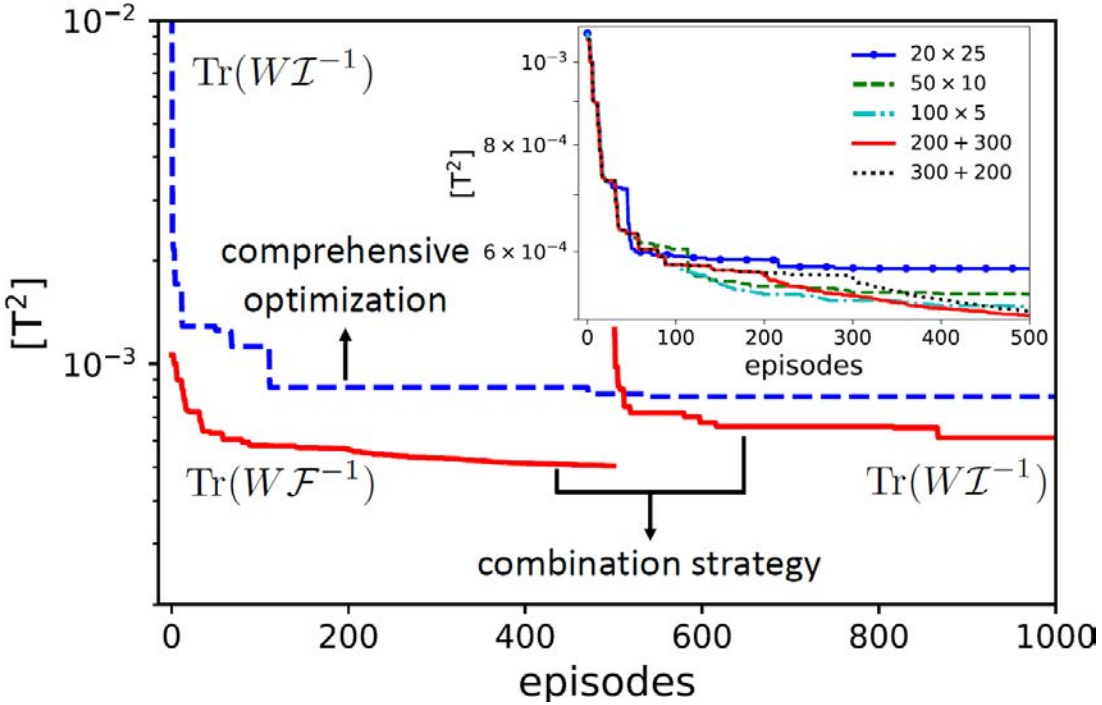
Comprehensive optimization

State and control optimization

```
kwargs = {"p_num":10, "psi0"=[], "ctrl0"=[],\
          "measurement0":[], "max_episode":1000,\
          "c":1.0, "cr":0.5, "seed":1234}
com = ComprehensiveOpt(savefile=False,method="DE",**kwargs)
com.dynamics(tspan, rho0, H0, dH, Hc, decay=decay,\
             ctrl_bound=[-0.2, 0.2], dyn_method="expm")
com.SC(target="QFIM", LDtype="SLD")
```

State, control, and measurement optimization

```
kwargs = {"p_num":10, "psi0"=[], "ctrl0"=[],\
          "measurement0":[], "max_episode":1000,\
          "c":1.0, "cr":0.5, "seed":1234}
com = ComprehensiveOpt(savefile=False,method="DE",**kwargs)
com.dynamics(tspan, rho0, H0, dH, Hc, decay=decay,\
             ctrl_bound=[-0.2, 0.2], dyn_method="expm")
com.SCM(target="QFIM", LDtype="SLD")
```



Adaptive measurement

FOP

unknown

tunable

Optimal point $\mathbf{x}_{\text{opt}} \longleftarrow \mathbf{x} + \mathbf{u}$

Bayes' rule $$p(\mathbf{x}, \mathbf{u}^{(n)}|y^{(n)}) = \frac{p(y^{(n)}|\mathbf{x}, \mathbf{u}^{(n)})p(\mathbf{x})}{\int p(y^{(n)}|\mathbf{x}, \mathbf{u}^{(n)})p(\mathbf{x})\mathrm{d}\mathbf{x}}$$

Estimated value $$\hat{\mathbf{x}}^{(n)} = \operatorname{argmax} p(\mathbf{x}, \mathbf{u}^{(n)}|y^{(n)})$$

Tunable value $$\mathbf{u}^{(n+1)} = \mathbf{x}_{\text{opt}} - \hat{\mathbf{x}}^{(n)}$$

MI

Optimization of the mutual information

$$I(\mathbf{u}) = \int \mathrm{p}(\mathbf{x}) \sum_{y} \mathrm{p}(y|\mathbf{x}, \mathbf{u}) \log_2 \left[\frac{\mathrm{p}(y|\mathbf{x}, \mathbf{u})}{\int \mathrm{p}(\mathbf{x})\mathrm{p}(y|\mathbf{x}, \mathbf{u})\mathrm{d}\mathbf{x}} \right] \mathrm{d}\mathbf{x}$$

Tunable value $$\mathbf{u}^{(n+1)} = \operatorname{argmax} I(\mathbf{u})$$

Adaptive measurement

```
apt = Adapt(x,p,rho0,method="FOP",savefile=False,\
            max_episode=1000,eps=1e-8)
apt.dynamics(tspan,H,dH,Hc=[],ctrl=[],decay=[],\
             dyn_method"expm")
apt.CFIM(M=[],W=[])
```

Adaptive measurement

Hamiltonian $H = \frac{\kappa\omega_0}{2}(\sigma_1 \cos x + \sigma_3 \sin x)$ Dynamics $\partial_t \rho = -i[H, \rho]$

Measurement $\{\Pi_1, \Pi_2, I - \Pi_1 - \Pi_2\}$ Initial state $|+\rangle$

$\Pi_1 = 0.85|00\rangle\langle 00| \; \Pi_2 = 0.1|++\rangle\langle ++|$

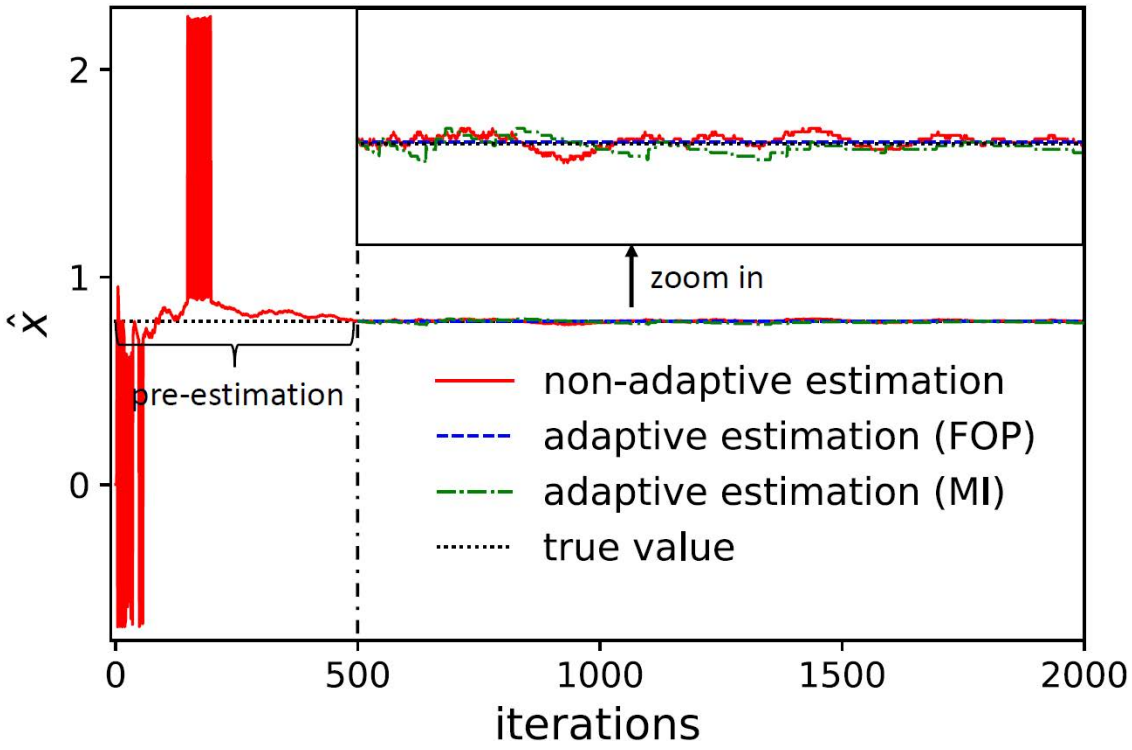
[ ]:
```
from quanestimation import *
from qutip import *
import numpy as np
import random
from scipy.integrate import simps

# initial state
rho0 = 0.5 * np.ones((2, 2))

# free Hamiltonian
kappa, omega0 = np.pi/2.0, 1.0
s1, s2, s3 = sigmax(), sigmay(), sigmaz()
s1, s2, s3 = s1.full(), s2.full(), s3.full()
H0_func = lambda x: 0.5*kappa*omega0*\
                    (s1*np.cos(x)+s3*np.sin(x))

# derivative of the free Hamiltonian on x
dH_func = lambda x: [
    0.5*kappa*omega0*(-s1*np.sin(x)+s3*np.cos(x))
]
```

[ ]:
```
# time span for the evolution
tspan = np.linspace(0., 1., 1000)
# parameterization
x = np.linspace(0., 0.5*np.pi, 1000)
rho = [
    np.zeros((len(rho0), len(rho0)), dtype=np.complex128)
    for i in range(len(x))
]
for i in range(len(x)):
    H0 = H0_func(x[i])
    dH = dH_func(x[i])
    dynamics = Lindblad(tspan, rho0, H0, dH)
    rho_tp, drho_tp = dynamics.expm()
    rho[i] = rho_tp[-1]

# measurement
M1 = 0.5 * np.ones((2, 2))
M2 = (identity(2) - M1).full()
M = [M1, M2]
```

Adaptive measurement

```
[ ]: # prior distribution
     p = (1.0/(x[-1]-x[0]))*np.ones(len(x))

     # Generation of the experimental results
     y = [ 0 if np.random.randn() > 0.5 else 1 for _ in range(500)]
```

```
[ ]: # Bayesian estimation
     pout, xout = Bayes([x], p, rho, y, M=M, estimator="MAP",
                        savefile=False)
     # generate H and dH
     H, dH = BayesInput([x], H0_func, dH_func, channel="dynamics")
     # adaptive estimation
     apt = Adapt([x], pout, rho0, savefile=False,
                 method="FOP", max_episode=100, eps=1e-8)
     apt.dynamics(tspan, H, dH)
     apt.CFIM (M=M, W=[])
```



Adaptive measurement

Online adaptive phase estimation

```
apt = Adapt_MZI(x,p,rho0)
apt.general()
apt.online(target="sharpness",output="phi")
```

Offline adaptive phase estimation

```
apt.offline(target="sharpness",method="DE",**kwargs)
```


D. W. Berry and H. M. Wiseman, Phys. Rev. Lett. **85**, 5098 (2000).

D. W. Berry, H. M. Wiseman, and J. K. Breslin, Phys. Rev. A **63**, 053804 (2001).

K. Rambhatla et al., Phys. Rev. Research **2**, 033078 (2020).

QuanEstimation: Please try it !

Python package:

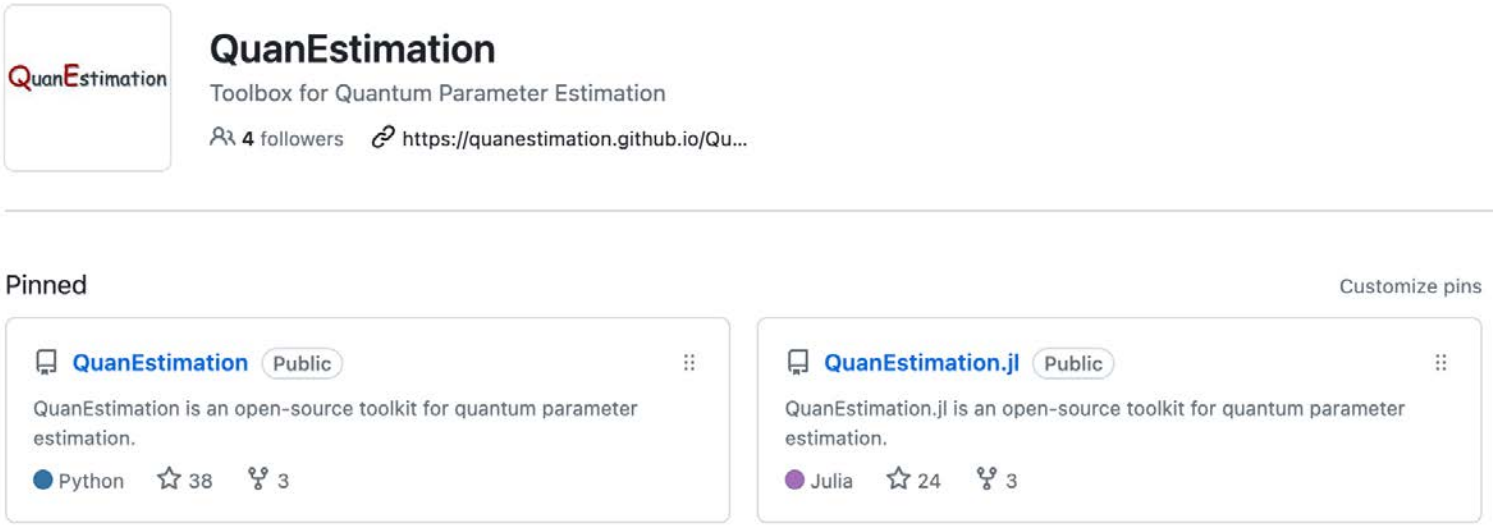
https://github.com/QuanEstimation/QuanEstimation

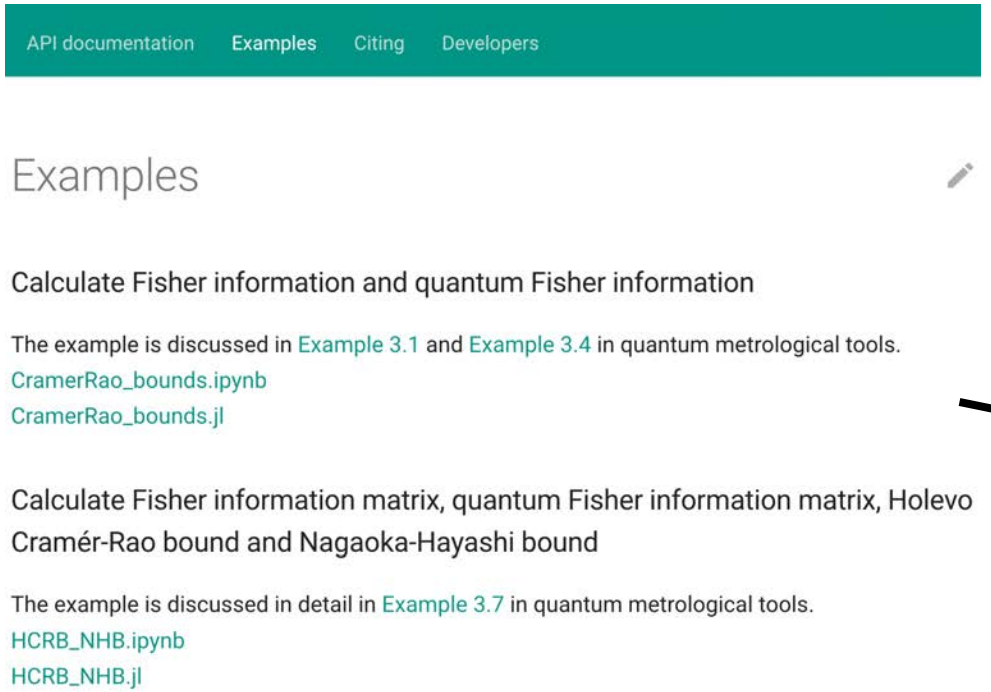
Julia package:

https://github.com/QuanEstimation/QuanEstimation.jl

Documentation:

https://quanestimation.github.io/QuanEstimation/





Tutorial

bilibili 余怀明 — QuanEstimation.jl: Julia与量子参数估计数值计算

Papers

[1] M. Zhang et al., Phys. Rev. Research **4**, 043057 (2022).
https://doi.org/10.1103/PhysRevResearch.4.043057

[2] H.-M. Yu. et al., in preparation.

Congratulations, this is the last page

Thank you