Quantum Fisher Information During Quantum Annealing in an Ising-Chain Sensor

Overview

We consider a many-body sensor based on an Ising chain. The unknown parameter is a transverse field amplitude h that couples to $\sum_j X_j$. The annealing Hamiltonian is

$$H_h(t) \;=\; A(t)\, H_d \;+\; B(t)\, h \sum_{j=1}^{N} X_j, \qquad H_d = J \sum_{j=1}^{N-1} Z_j Z_{j+1}\,, \tag{1}$$

with open boundary conditions. The system starts in the ground state $|\psi_0\rangle$ of H_d and evolves under the time-ordered unitary $U_h(t,0)$ generated by $H_h(t)$.

For a *pure* output state $|\psi(T;h)\rangle = U_h(T,0)\,|\psi_0\rangle$, the quantum Fisher information (QFI) for estimating h can be written without finite differences using the *generator* $G_h(T)$:

$$G_h(T) = \int_0^T U_h^\dagger(t,0)\, \partial_h H_h(t)\, U_h(t,0)\, dt, \tag{2}$$

$$F_Q(T) = 4\Big(\langle\psi_0|G_h(T)^2|\psi_0\rangle - |\,\langle\psi_0|G_h(T)|\psi_0\rangle|^2 \Big)\,. \tag{3}$$

Because $\partial_h H_h(t) = B(t)\sum_j X_j$, the QFI naturally accumulates information only when $B(t) \neq 0$, and the interaction picture in Eq. (2) accounts for non-commutativity at different times. Numerically, we approximate the integral by a Riemann sum with time step Δt:

$$G_h(T) \approx \sum_{k=0}^{K-1} \left[U_h^\dagger(t_k,0)\, B(t_k) \sum_j X_j\, U_h(t_k,0) \right] \Delta t, \qquad t_k = k\,\Delta t, \quad K\Delta t = T. \tag{4}$$

This avoids explicit derivatives of wavefunctions and is stable even when $H_h(t)$ at different times does not commute.

Schedules. We use a simple monotone schedule $A(t) = 1 - s(t)$, $B(t) = s(t)$ with $s(t) = (t/T)^\alpha$ ($\alpha \geq 1$ gives slower turn-on).

What the code does (summary)

1. Build Pauli operators on N spins and H_d, $X_{\text{sum}} = \sum_j X_j$.
2. Find the ground state $|\psi_0\rangle$ of H_d by exact diagonalization.
3. Evolve from $t = 0$ to T with small steps Δt using midpoint Hamiltonians.
4. Accumulate the generator $G_h(T)$ via Eq. (2).
5. Compute $F_Q(T)$ via Eq. (3).
6. Optionally, compute $F_Q(t)$ *during* the anneal for a list of intermediate times.

Python reference implementation

Save the following as `qa_qfi.py` and run with `python qa_qfi.py`. Requires `numpy` and `scipy`.

```
import numpy as np
from numpy import kron
from scipy.linalg import eigh, expm

# ---------- Utilities ----------
def pauli_mats():
    I = np.array([[1,0],[0,1]], complex)
    X = np.array([[0,1],[1,0]], complex)
    Y = np.array([[0,-1j],[1j,0]], complex)
    Z = np.array([[1,0],[0,-1]], complex)
    return I, X, Y, Z

def op_on_site(single_op, site, N):
    """Return N-qubit operator with 'single_op' at 'site' (0-based), identity
        elsewhere."""
    I, _, _, _ = pauli_mats()
    ops = [I]*N
    ops[site] = single_op
    M = ops[0]
    for k in range(1, N):
        M = kron(M, ops[k])
    return M

def sum_two_body_ZZ(N, J=1.0, open_chain=True):
    """H_d = J sum_j Z_j Z_{j+1} (open or periodic)."""
    _, _, _, Z = pauli_mats()
    dim = 2**N
    H = np.zeros((dim, dim), dtype=complex)
    last = N-1
    pairs = [(j, j+1) for j in range(N-1)]
    if not open_chain:
        pairs.append((last, 0))
    for (j,k) in pairs:
        H += J * (op_on_site(Z, j, N) @ op_on_site(Z, k, N))
    return H

def sum_one_body_X(N):
    """X_sum = sum_j X_j"""
    _, X, _, _ = pauli_mats()
    dim = 2**N
    Xsum = np.zeros((dim, dim), dtype=complex)
    for j in range(N):
        Xsum += op_on_site(X, j, N)
    return Xsum

def ground_state(H):
    """Return (E0, |psi0>) for the smallest eigenvalue of H."""
    E, V = eigh((H + H.conj().T)/2) # Hermitize for numerical stability
    return E[0], V[:, 0]

# ---------- Annealing schedule ----------
def s_of_t(t, T, alpha=1.0):
    s = (t / T) ** alpha
    return min(max(s, 0.0), 1.0)
```

```
def A_of_t(t, T, alpha=1.0): return 1.0 - s_of_t(t, T, alpha)
def B_of_t(t, T, alpha=1.0): return s_of_t(t, T, alpha)

# ---------- Core: QFI during annealing ----------
def qfi_during_anneal(N=4, J=1.0, h=0.2, T=5.0, K=1000, alpha=2.0,
                      open_chain=True, return_trace=True):
    """
    Compute F_Q(T) for parameter h during QA with H(t)=A(t) Hd + B(t) h X_sum.
    If return_trace=True, also return F_Q at intermediate times.
    """
    Hd = sum_two_body_ZZ(N, J=J, open_chain=open_chain)
    Xsum = sum_one_body_X(N)
    E0, psi0 = ground_state(Hd)

    dt = T / K
    dim = 2**N
    U = np.eye(dim, dtype=complex) # U(t,0)
    G = np.zeros((dim, dim), dtype=complex)

    # optional trace of F_Q(t)
    times = []
    FQs = []

    for k in range(K):
        t = k * dt
        # Midpoint Hamiltonian for better accuracy
        tm = t + 0.5*dt
        Hmid = A_of_t(tm, T, alpha) * Hd + B_of_t(tm, T, alpha) * (h * Xsum)
        # Propagate U -> U_next
        Ustep = expm(-1j * Hmid * dt)
        # Accumulate G using U(t,0) at the *start* of the interval
        dHdh = B_of_t(t, T, alpha) * Xsum
        G += (U.conj().T @ dHdh @ U) * dt
        # Update U
        U = Ustep @ U

        if return_trace and ( (k % max(1, K//200)) == 0 ):
            # Partial G accumulation approximates G(t)
            mean = np.vdot(psi0, G @ psi0)
            var = np.vdot(psi0, (G @ G) @ psi0) - (mean * np.conj(mean))
            FQ = 4.0 * np.real(var)
            times.append(t)
            FQs.append(FQ)

    # Final QFI at T
    mean = np.vdot(psi0, G @ psi0)
    var = np.vdot(psi0, (G @ G) @ psi0) - (mean * np.conj(mean))
    FQ_T = 4.0 * np.real(var)

    if return_trace:
        return FQ_T, (np.array(times), np.array(FQs))
    else:
        return FQ_T, None

if __name__ == "__main__":
    # --- User parameters ---
    N = 4 # number of spins (2^N Hilbert dimension)
    J = 1.0 # Ising coupling
```

```
h = 0.2 # unknown field amplitude (point where QFI is evaluated)
T = 5.0 # total anneal time
K = 2000 # time steps; increase for accuracy
alpha = 2.0 # schedule exponent; 1.0 = linear, >1 slows turn-on of B(t)

FQ_T, trace = qfi_during_anneal(N=N, J=J, h=h, T=T, K=K, alpha=alpha, return_trace
    =True)
print(f"Final QFI at T={T:.3f}: {FQ_T:.8f}")

# Optionally: write the time trace to CSV for plotting elsewhere
if trace is not None:
    times, FQs = trace
    data = np.column_stack([times, FQs])
    np.savetxt("qfi_trace.csv", data, delimiter=",", header="t,FQ(t)", comments="")
    print("Saved QFI time trace to qfi_trace.csv")
```

Usage notes and extensions

- **Scaling.** The code uses exact matrices of size $2^N \times 2^N$. For $N \gtrsim 10$ this becomes heavy; consider matrix-free Krylov solvers or tensor networks (TEBD/MPS) for larger N.
- **Schedules.** Replace $A(t), B(t)$ as needed (e.g. pause windows, reverse anneals, shortcuts to adiabaticity).
- **Noise.** The formula above is for unitary dynamics. With Markovian noise, pure-state QFI no longer applies; use SLD/Lindbladian approaches or compute the Bures metric from the evolved density matrix.
- **Comparisons.** To benchmark "before/after" QA, compare the final $F_Q(T)$ to: (i) zero-anneal case $T = 0$ (i.e. directly measuring in $|\psi_0\rangle$), and (ii) adiabatic limit (increase T) which approaches the ground state of H_c.