Quandary for Spin Chain Simulations

Jan 2023

Stefanie Guenther

CASC
Center for Applied Scientific Computing

LLNL-PRES- xxxxxx
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Quandary simulates the dynamics of general composite open and closed quantum systems

- Composite of Q (coupled) subsystems, each with n_q finite energy levels $\Longrightarrow N = \prod_q n_q$
- **Closed** systems (unitary evolution):
  Quandary solves **Schroedinger's equation** for the state vector $\boldsymbol{\psi} \in \mathbb{C}^{\boldsymbol{N}}$

$$\boxed{\frac{d\,\psi}{d\,t} = -iH(t)\psi}$$

- **Open** systems to account for decoherence:
  Quandary solves **Lindblad's master equation** for the density matrix $\boldsymbol{\rho} \in \mathbb{C}^{\boldsymbol{N\times N}}$

$$\boxed{\frac{d\,\rho}{d\,t} = -i\big(H(t)\rho - \rho H(t)\big) + L(\rho)}$$

Decay and Dephasing: $L(\rho) = \sum_{k=1,2} L_k\rho L_k^\dagger - \frac{1}{2}(L_k^\dagger L_k\rho + \rho L_k^\dagger L_k)$

- Symplectic time-stepping scheme to evolve initial states at time t=0 to a final time t=T

Default system and control Hamiltonian to model superconducting quantum devices

$$\boxed{H(t) = H_{sys} + H_c(t)}$$

Lowering operator for qubit j:

$$\boldsymbol{a_j} = I \otimes \cdots \otimes \hat{\boldsymbol{a}} \otimes \cdots \otimes I$$

with $\hat{\boldsymbol{a}} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$

- System Hamiltonian model:

$$H_{sys} = \sum_{j=1}^{Q} \omega_j \boldsymbol{a}_j^\dagger \boldsymbol{a}_j - \frac{\xi_j}{2} \boldsymbol{a}_j^\dagger \boldsymbol{a}_j^\dagger \boldsymbol{a}_j \boldsymbol{a}_j - \sum_{i>j} \xi_{ji} \boldsymbol{a}_j^\dagger \boldsymbol{a}_j^\dagger \boldsymbol{a}_i \boldsymbol{a}_i - J_{ji}\left(\boldsymbol{a}_j^\dagger \boldsymbol{a}_i + \boldsymbol{a}_j \boldsymbol{a}_i^\dagger\right)$$

Qubit frequencies, anharmonicity

Coupling: ZZ- and/or Jaynes-Cummings

- Control Hamiltonian models the action of external pulses

$$H_c(t) = \sum_{j=1}^{Q} f_j(t)\left(\boldsymbol{a}_j + \boldsymbol{a}_j^\dagger\right)$$

External pulses can be either specified (simulation), or optimized for to reach a desired system behavior

Python interface for general Hamiltonian systems

- Quandary can simulate and optimize with **arbitrary user-defined Hamiltonian systems** $\mathrm{H(t)} = \mathrm{H_{sys}} + \mathrm{H_c(t)} \in \mathbb{C}^{N\times N}$

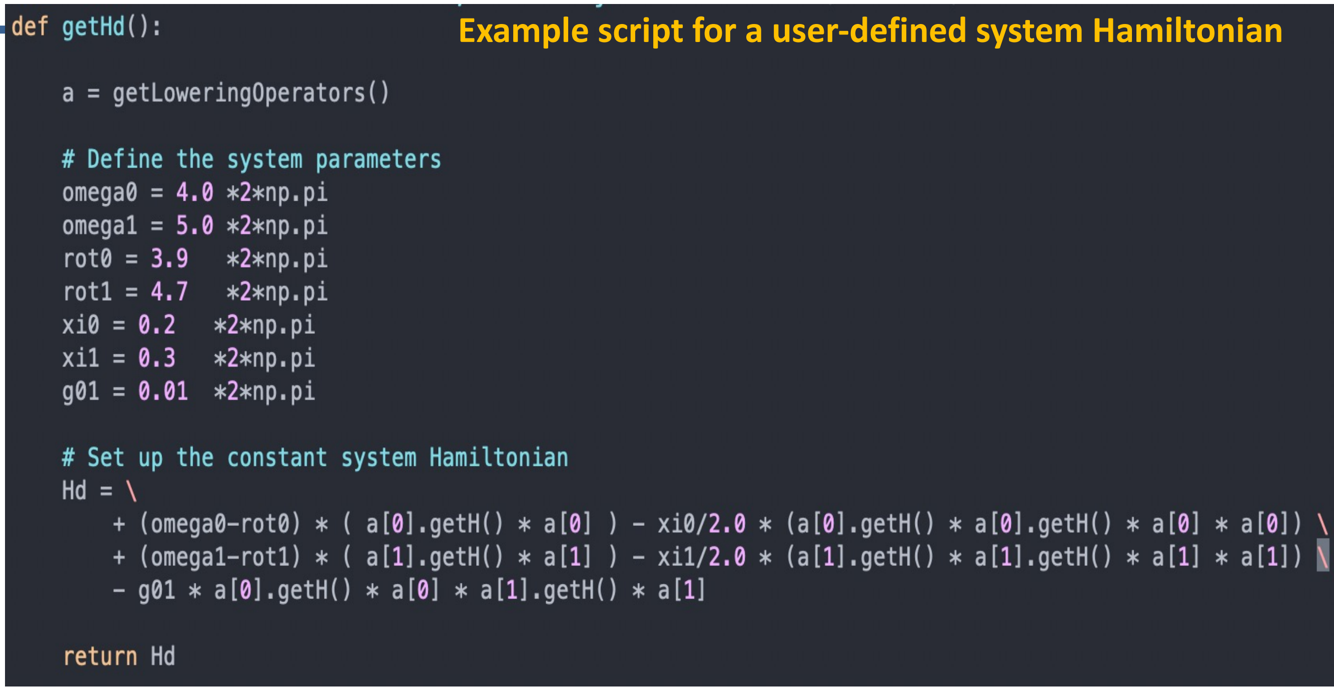
$\boxed{\frac{d\,\psi}{d\,t} = -iH(t)\psi}$ or $\boxed{\frac{d\,\rho}{d\,t} = -i\big(H(t)\rho - \rho H(t)\big) + L(\rho)}$

- Multiple user-defined control operators and transfer functions can be defined

Control operators

$$H_c(t) = \sum_{c=1,2,\ldots} u\big(p(t)\big)\boldsymbol{H_c^{Re}} + iv\big(q(t)\big)\boldsymbol{H_c^{Im}}$$

Transfer functions

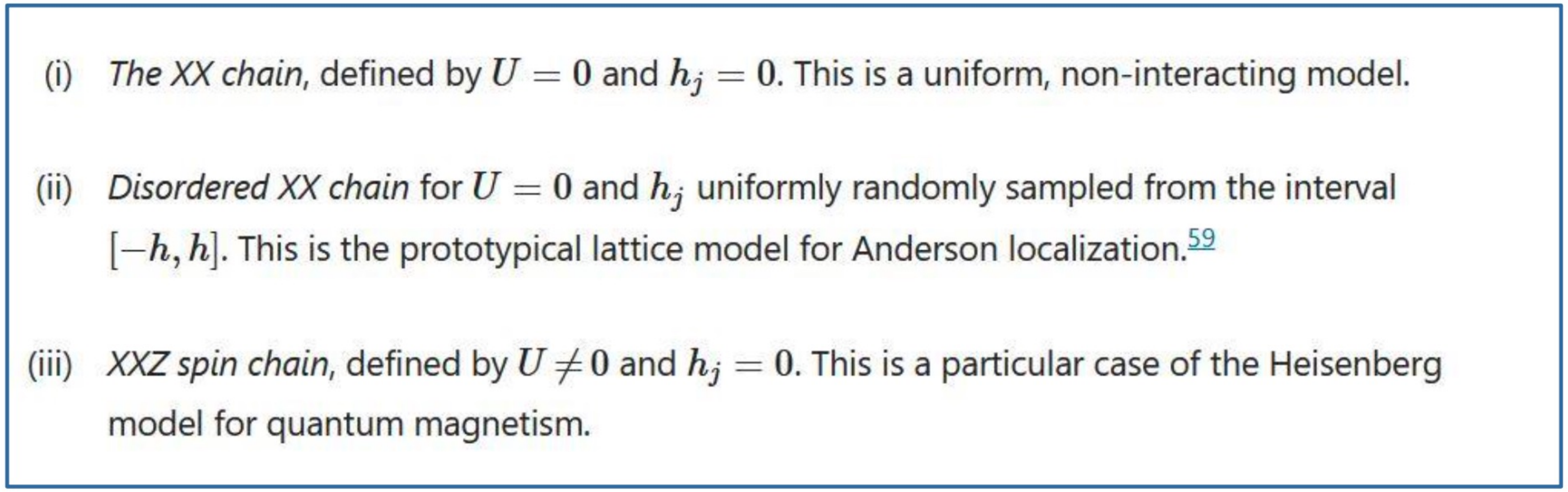


Example script for a user-defined system Hamiltonian

```
def getHd():

    a = getLoweringOperators()

    # Define the system parameters
    omega0 = 4.0 *2*np.pi
    omega1 = 5.0 *2*np.pi
    rot0 = 3.9   *2*np.pi
    rot1 = 4.7   *2*np.pi
    xi0 = 0.2   *2*np.pi
    xi1 = 0.3   *2*np.pi
    g01 = 0.01  *2*np.pi

    # Set up the constant system Hamiltonian
    Hd = \
        + (omega0-rot0) * ( a[0].getH() * a[0] ) - xi0/2.0 * (a[0].getH() * a[0].getH() * a[0] * a[0]) \
        + (omega1-rot1) * ( a[1].getH() * a[1] ) - xi1/2.0 * (a[1].getH() * a[1].getH() * a[1] * a[1]) \
        - g01 * a[0].getH() * a[0] * a[1].getH() * a[1]

    return Hd
```

Quantum Spin Chain Dynamics

$$\hat{H} = -J\sum_{j=1}^{N-1}\left(\hat{\sigma}_j^x\hat{\sigma}_{j+1}^x + \hat{\sigma}_j^y\hat{\sigma}_{j+1}^y\right) + U\sum_{j=1}^{N-1}\hat{\sigma}_j^z\hat{\sigma}_{j+1}^z + \sum_{j=1}^{N} h_j\hat{\sigma}_j^z,$$



(i) *The XX chain*, defined by $U = 0$ and $h_j = 0$. This is a uniform, non-interacting model.

(ii) *Disordered XX chain* for $U = 0$ and h_j uniformly randomly sampled from the interval $[-h, h]$. This is the prototypical lattice model for Anderson localization.[59]

(iii) *XXZ spin chain*, defined by $U \neq 0$ and $h_j = 0$. This is a particular case of the Heisenberg model for quantum magnetism.

Initial state at t=0:

Domain wall

$$|\cdots\downarrow\downarrow\downarrow\uparrow\uparrow\uparrow\cdots\rangle, \longrightarrow$$

Neel

$$|\cdots\uparrow\downarrow\uparrow\downarrow\uparrow\cdots\rangle \longrightarrow$$

Observables / Diagnostics:

– Local magnetization: $M_j(t) = \langle\psi(t)|\sigma_j^z|\psi(t)\rangle$

– Spreading of the domain wall: $N_{half}(t) = \sum_{j=1}^{N/2}\frac{1}{2}\langle\psi(t)|\sigma_j^z + I\,|\psi(t)\rangle$

Map 1D spin chain Pauli algebra to creation/annihilation algebra as used in Quandary

$$\hat{H} = \underbrace{-J\sum_{j=1}^{N-1}\left(\hat\sigma_j^x\hat\sigma_{j+1}^x + \hat\sigma_j^y\hat\sigma_{j+1}^y\right)}_{I.} + \underbrace{U\sum_{j=1}^{N-1}\hat\sigma_j^z\hat\sigma_{j+1}^z}_{II.} + \underbrace{\sum_{j=1}^{N} h_j\hat\sigma_j^z}_{III.},$$

- Lowering operator $a = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$
- Pauli matrices:

$$\sigma^x = (a + a^\dagger)$$
$$\sigma^y = -i(a - a^\dagger)$$
$$\sigma^z = -2a^\dagger a + I$$

I. $-J\left(\sigma_j^x\sigma_{j+1}^x + \sigma_j^y\sigma_{j+1}^y\right) = -2J\left(a_j a_{j+1}^\dagger + a_j^\dagger a_{j+1}\right)$

II. $U\sigma_j^z\sigma_{j+1}^z = U\left(4a_j^\dagger a_j a_{j+1}^\dagger a_{j+1} - 2a_j^\dagger a_j - 2a_{j+1}^\dagger a_{j+1} + I\right)$

III. $h_j\sigma_j^z = h_j\left(-2a_j^\dagger a_j + I\right)$

$$\boxed{\Rightarrow H = -2\sum_{j=1}^{N-1} J\left(\boldsymbol{a_j a_{j+1}^\dagger} + \boldsymbol{a_j^\dagger a_{j+1}}\right) + 2\sum_{j=1}^{N-1} 2U\boldsymbol{a_j^\dagger a_j a_{j+1}^\dagger a_{j+1}} - 2\sum_{j=2}^{N-1}(h_j + 2U)\boldsymbol{a_j^\dagger a_j} + \sum_{j=2}^{N-1}(h_j + U)I + (h_1 + h_N)I}$$

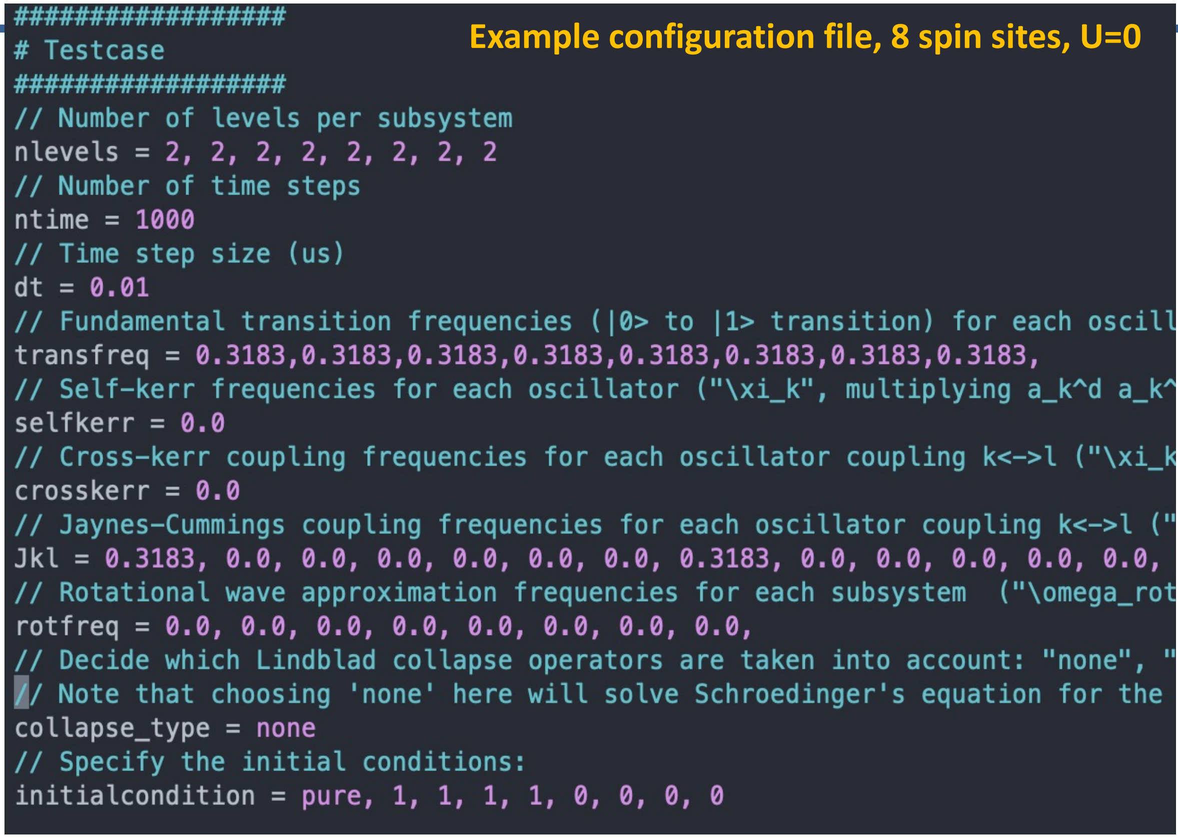
Compare to Quandary model: $H_{sys} = \sum_{j=1}^{N} \omega_j \boldsymbol{a_j^\dagger a_j} - \frac{\xi_j}{2} a_j^\dagger a_j\left(a_j^\dagger a_j - I\right) + \sum_{i>j} -\xi_{ji}\boldsymbol{a_j^\dagger a_j a_i^\dagger a_i} + J_{ij}\left(\boldsymbol{a_i^\dagger a_j} + \boldsymbol{a_i a_j^\dagger}\right)$

Quandary execution to simulate spin chain dynamics

$$H_{spinchain} = \sum_{j=1}^{N-1} (h_j + 2U) \boldsymbol{a}_j^\dagger \boldsymbol{a}_j - \sum_{j=1}^{N-1} 2U \boldsymbol{a}_j^\dagger \boldsymbol{a}_j \boldsymbol{a}_{j+1}^\dagger \boldsymbol{a}_{j+1} + \sum_{j=1}^{N-1} J \left(\boldsymbol{a}_j \boldsymbol{a}_{j+1}^\dagger + \boldsymbol{a}_j^\dagger \boldsymbol{a}_{j+1} \right)$$

$$H_{Quandary} = \underbrace{\sum_{j=1}^{N} \omega_j a_j^\dagger a_j}_{\text{“transfreq”}} + \underbrace{\sum_{i>j} - \xi_{ji} a_j^\dagger a_j a_i^\dagger a_i}_{\text{“crossker”}} + \underbrace{\sum_{i>j} J_{ij} (a_i^\dagger a_j + a_i a_j^\dagger)}_{\text{“Jkl”}}$$

- Quandary (C++) is executed through a configuration file
  `>./main config_file.cfg`
- Configuration options for spin chain simulation:
  - ➢ Set number of sites and levels: `nlevels = 2, 2, …, 2`
  - ➢ Set transition frequencies: `transfreq = h1+2U, …, hN+2U`
  - ➢ Set Jaynes-Cummings coupling: `Jkl = J, 0, …, J, 0, …, J`
  - ➢ Set crossker coupling: `crossker = 2U, 0, …, 2U, 0, …, 2U`
  - ➢ Set initial condition: `initialstate = pure, 1,1,1,…,0,0,0…`
  - ➢ Turn off optimization: `optim_target = none`
  - ➢ No controls: `optim_init_ampl = 0.0`

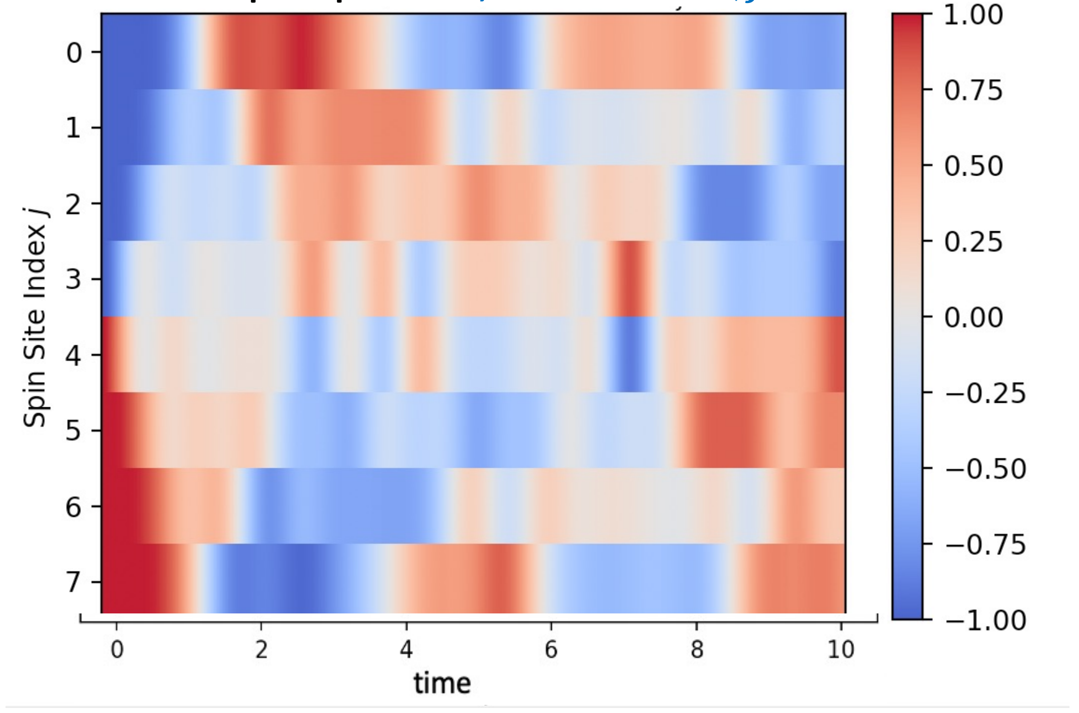

Example configuration file, 8 spin sites, U=0

```
##################
# Testcase
##################
// Number of levels per subsystem
nlevels = 2, 2, 2, 2, 2, 2, 2, 2
// Number of time steps
ntime = 1000
// Time step size (us)
dt = 0.01
// Fundamental transition frequencies (|0> to |1> transition) for each oscill
transfreq = 0.3183,0.3183,0.3183,0.3183,0.3183,0.3183,0.3183,0.3183,
// Self-kerr frequencies for each oscillator ("\xi_k", multiplying a_k^d a_k^
selfkerr = 0.0
// Cross-kerr coupling frequencies for each oscillator coupling k<->l ("\xi_k
crosskerr = 0.0
// Jaynes-Cummings coupling frequencies for each oscillator coupling k<->l ("
Jkl = 0.3183, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3183, 0.0, 0.0, 0.0, 0.0, 0.0,
// Rotational wave approximation frequencies for each subsystem  ("\omega_rot
rotfreq = 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
// Decide which Lindblad collapse operators are taken into account: "none", "
// Note that choosing 'none' here will solve Schroedinger's equation for the
collapse_type = none
// Specify the initial conditions:
initialcondition = pure, 1, 1, 1, 1, 0, 0, 0, 0
```

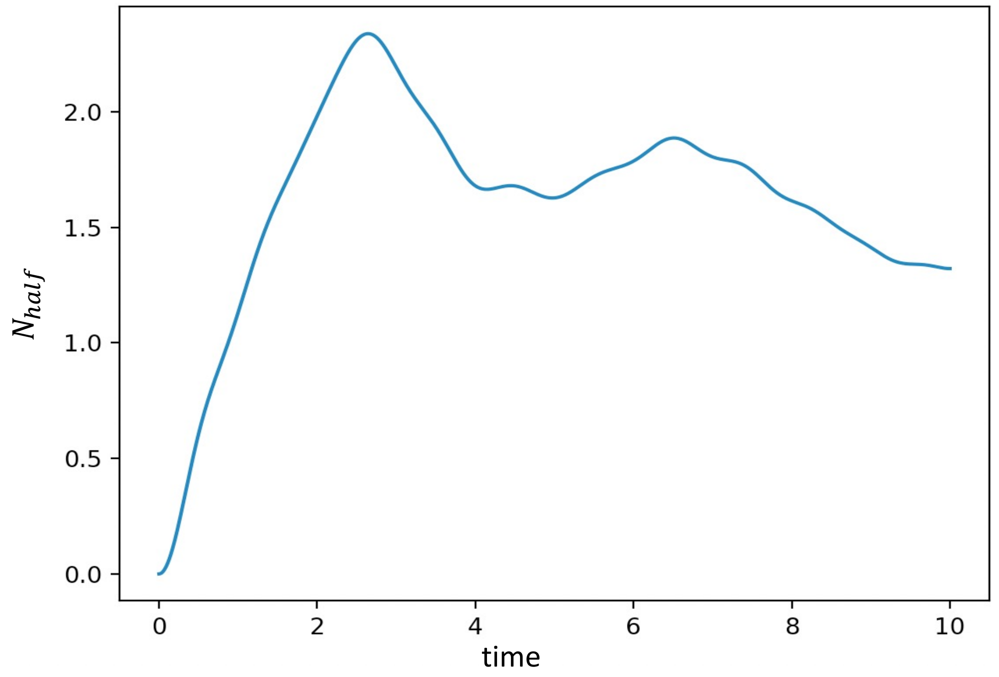
Example: Disordered XX model (U=0)

N=8 spin sites

Heat map of Spin Chain, constant $h = 1, J = 1$



Averaged N_{half} for $h \sim U([-1,1])$, constant $J = 1$



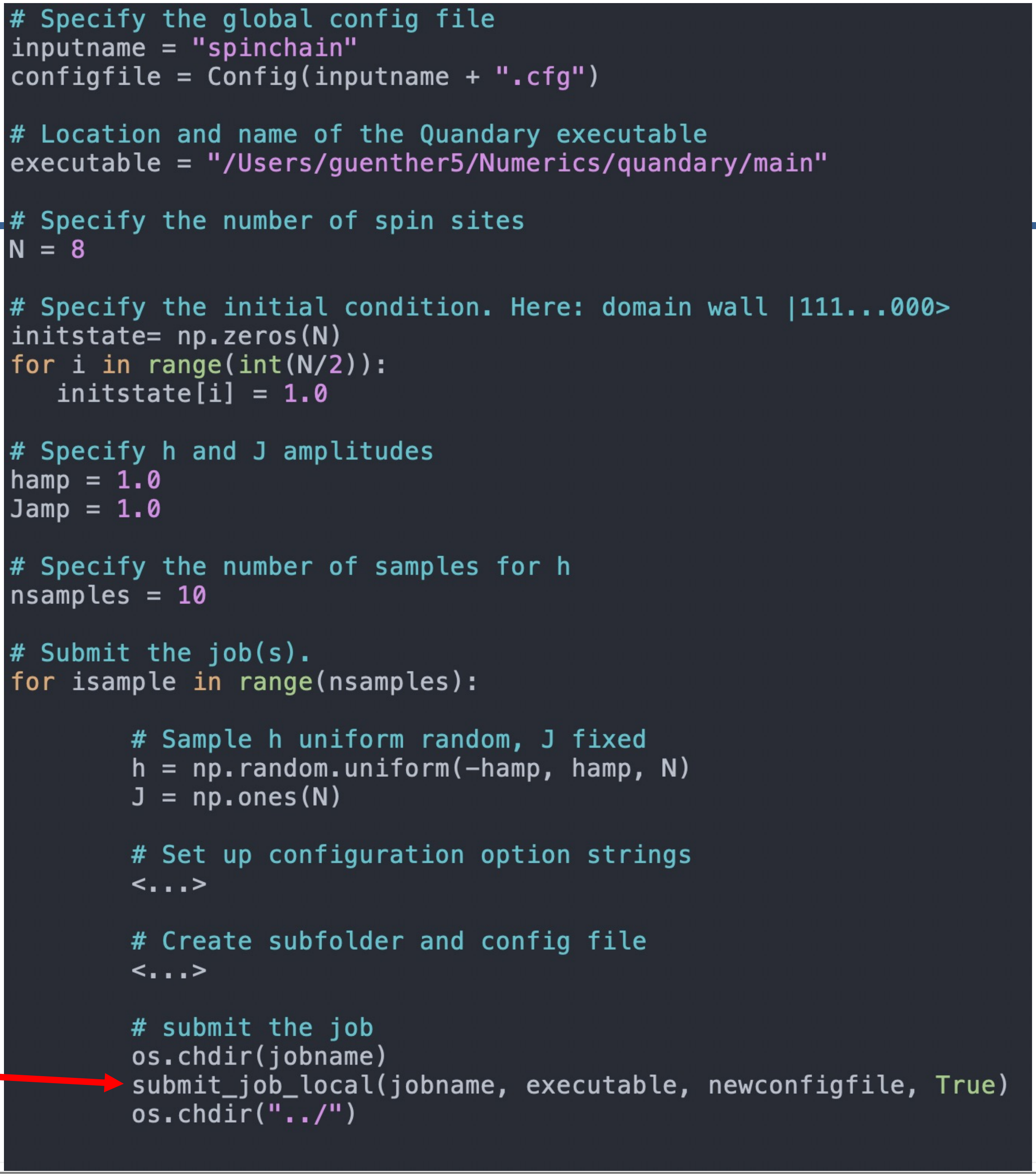
Python scripts to execute multiple simulations and gather results

`run_disorderedXX.py`

- Sample $h \sim U([-1,1])$
- For each sample:
  - Dump Quandary configuration files into subfolders
  - Execute Quandary simulation, or submit batch job on LC

`plotresults_disorderedXX.py`

- Gather results from each subfolder
- Plot



```
# Specify the global config file
inputname = "spinchain"
configfile = Config(inputname + ".cfg")

# Location and name of the Quandary executable
executable = "/Users/guenther5/Numerics/quandary/main"

# Specify the number of spin sites
N = 8

# Specify the initial condition. Here: domain wall |111...000>
initstate= np.zeros(N)
for i in range(int(N/2)):
    initstate[i] = 1.0

# Specify h and J amplitudes
hamp = 1.0
Jamp = 1.0

# Specify the number of samples for h
nsamples = 10

# Submit the job(s).
for isample in range(nsamples):

        # Sample h uniform random, J fixed
        h = np.random.uniform(-hamp, hamp, N)
        J = np.ones(N)

        # Set up configuration option strings
        <...>

        # Create subfolder and config file
        <...>

        # submit the job
        os.chdir(jobname)
        submit_job_local(jobname, executable, newconfigfile, True)
        os.chdir("../")
```

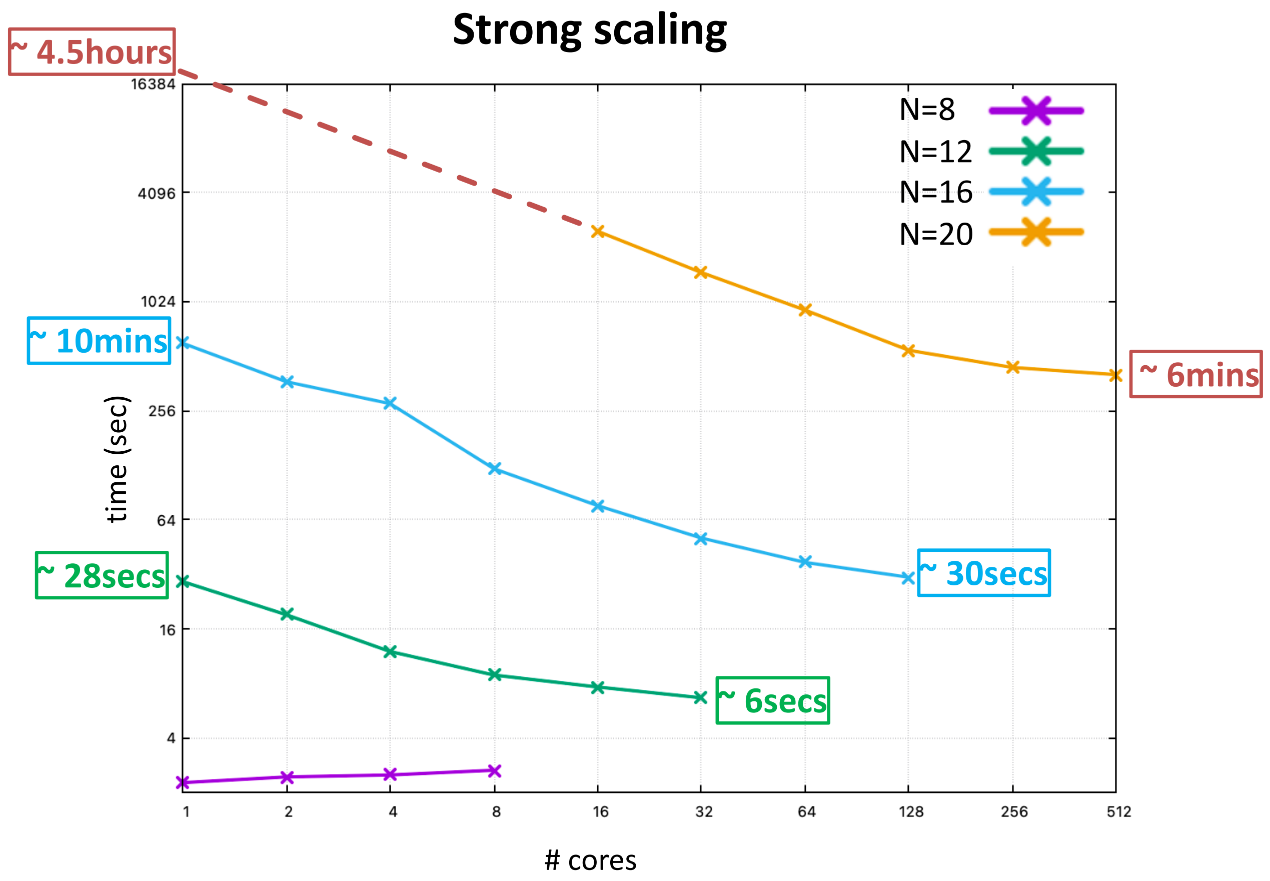
Run locally, or submit jobs on LC

N	# time steps	# cores
8	1000	
12	1500	[1,…32]
16	2000	[1,…,128]
20	2500	[16,…,512]

Parallel scaling on LC Quartz

Increase number of spin sites:

- $\dim(H_N) = 2^N$
- T=[0,10]
- Number of time-steps scales with $O(N)$
- Schroedinger solver



CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory