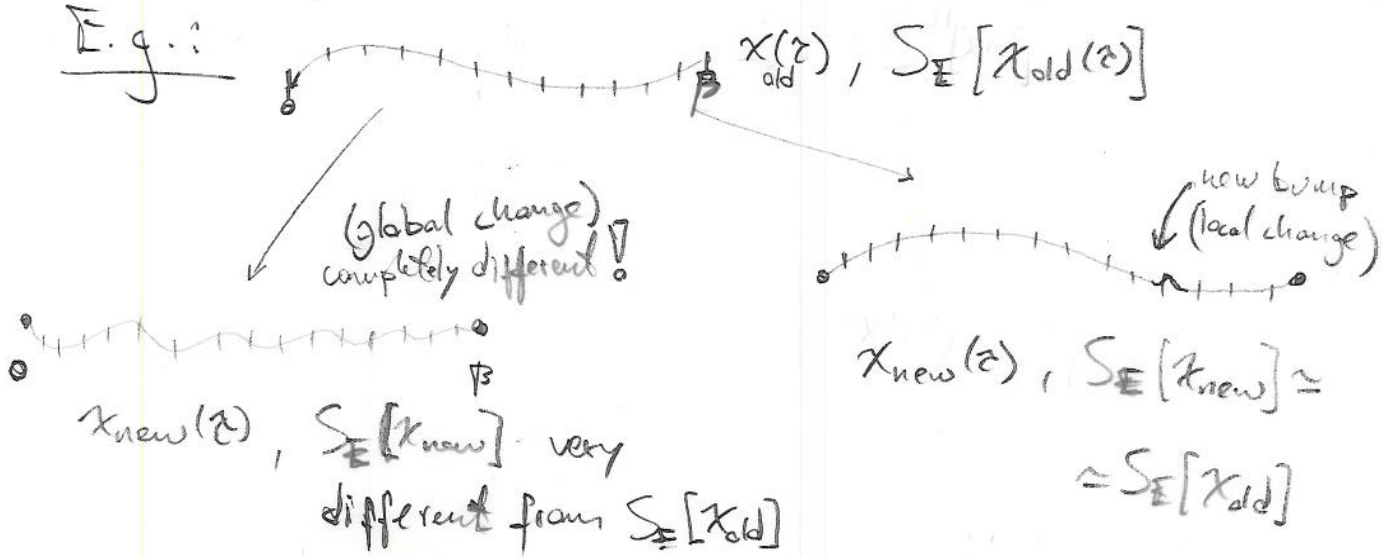
Quick recap. from last time

We know that

ⓐ We can generate samples according to any desired probability measure $P[x(\tau)]$ using the Metropolis algorithm, as long as $P \geq 0$ and $\int_{x(0)=x(\beta)} \mathcal{D}x \, P[x(\tau)] = 1$. (For us $P[x] = e^{-S_E[x]}$)

ⓑ When the dimensionality of the integral is too large, or the action too "complicated" (non-local, non-linear, etc.), it becomes harder and harder to explore configuration space efficiently. This is particularly true for fermions.

E.g.:



- How do we make global changes, and explore config. space efficiently, but maintaining a high acceptance rate?

Answer on the next page!

Molecular dynamics (MD)

In the form used regularly in lattice QCD applications, the formalism of molecular dynamics involves introducing an auxiliary momentum variable we shall call $\pi(\tau)$, as follows:

$$Z = \int_{x(0)=x(\beta)} \mathcal{D}x \; e^{-S_E[x(\tau)]} = \underbrace{\int \mathcal{D}\pi \; e^{-\int_0^\beta d\tau \frac{\pi^2(\tau)}{2}}}_{\text{decoupled factor!}} \int \mathcal{D}x \, e^{-S_E[x(\tau)]} =$$

$$= \int \mathcal{D}\pi \, \mathcal{D}x \; e^{-\mathcal{H}[\pi, x]}$$

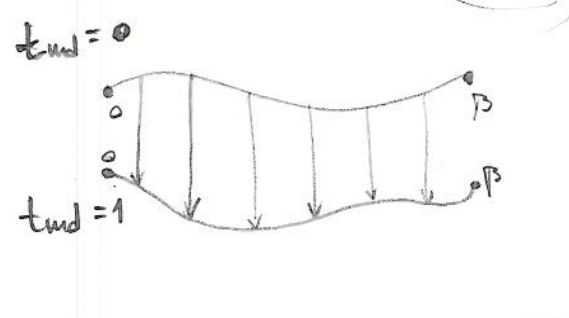
where $\mathcal{H} = \int_0^\beta d\tau \frac{\pi^2(\tau)}{2} + S_E[x(\tau)]$

Because $\mathcal{H}$ is quadratic in π we can sample π using a gaussian RNG. But we still have S_E ...

- <u>Idea</u>: Define equations of motion in a fictitious molecular dynamics time t_{md}

$$\frac{\partial}{\partial t_{md}} \begin{cases} \dot{x}(\tau) = \dfrac{\delta \mathcal{H}}{\delta \pi(\tau)} = \pi(\tau) \\ \dot{\pi}(\tau) = -\dfrac{\delta \mathcal{H}}{\delta x(\tau)} = -\dfrac{\delta S_E}{\delta x(\tau)} \end{cases}$$

functional derivative!



→ • If the action is fully local we have N_τ decoupled equations

• If the action has τ derivatives the eqns are coupled.
(or any derivatives in the case of fields)

→ The more non-local the action is, the more complicated the coupling b/w the eqns of motion.

• We need initial conditions:

- Gaussian momenta Π (Refreshed every so often in the MD evolution.)
- Any random config for X (Never refreshed, evolves according to the MD equations.)

With these equations of motion and initial conditions one is guaranteed to generate configurations of X according to the desired probability.

• But... there is a problem: How do we integrate the equations of motion?

In practice we introduce a finite time step Δt_{md}, and this introduces a systematic error.

Inversion strategies

Some of the most commonly used algorithms to simulate fermions involve the inversion of the "fermion matrix" M_{ij}

schematically)

$$S_E = \int dx \; \psi^+ M \psi$$

Grassmann variables.

- Why an inversion? Where did the inverse come from?

Recall that we saw in class that

$$\int D\psi^+ D\psi \; e^{-\int dx \, \psi^+ M \psi} = \mathrm{Det}\, M$$

Another way of writing $\mathrm{Det}\, M$ is as follows.

Let U diagonalize M, w/ $U^+ = U^{-1}$

$$U^+ M U = D = \underset{\text{matrix}}{\text{diagonal}} \Rightarrow \boxed{\mathrm{Det}\, M} = \mathrm{Det}\, U^+ M U = \mathrm{Det}\, D =$$

$$= \mathrm{diag}(\lambda_1, \dots, \lambda_N) \qquad = \prod_j \lambda_j = \exp\left\{\log\left[\prod_j \lambda_j\right]\right\} =$$

$$= \exp\left[\sum_j \log \lambda_j\right] =$$

$$= \exp\left[\mathrm{Tr} \log D\right] =$$

$$\boxed{= \exp\left[\mathrm{Tr} \log M\right]}$$

$$\mathrm{Tr}\left[f(M)\right] = \mathrm{Tr}\left[\sum_n a_n M^n\right]$$

$$= \mathrm{Tr}\left[\sum_n a_n \underbrace{U^+ M^n U}_{D^n}\right]$$

$$= \mathrm{Tr}\left[f(D)\right]$$

Now let us imagine that M depends on a parameter λ (e.g a coupling constant, the mass of the particles, etc.), and consider the following:

$$\frac{\partial(\det M)}{\partial\lambda} = \frac{\partial}{\partial\lambda}\left\{\exp\left[\mathrm{Tr}\log M\right]\right\} = \underbrace{\exp\left[\mathrm{Tr}\log M\right]}_{\det M} \times \mathrm{Tr}\left[M^{-1}\cdot\frac{\partial M}{\partial\lambda}\right]$$

$$= \det M \times \mathrm{Tr}\left[M^{-1}\frac{\partial M}{\partial\lambda}\right]$$

Aha! That's why the inverse is needed!

- Knowing how the fermion determinant varies requires knowing M^{-1}
- Also, as you saw in the problem sets, M^{-1} is a certain kind of "propagator", but we will get back to that later.

- <u>How do we invert M efficiently?</u>

There are various methods and various classifications of those methods. One of the most important distinctions involves the notions of

- <u>dense matrices</u> $\longrightarrow$ mostly non-zeros, not necessarily following any particular pattern (although they might)

vs.

- <u>sparse matrices</u> $\longrightarrow$ mostly zeros, often non-zeros arranged in a peculiar way (see next page).

In fact you already saw an example that looked like this:

$$M_0 = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ & -1 & 2 & -1 & 0 & \\ & & \ddots & \ddots & \ddots & \\ -1 & 0 & & & 0 & \\ & & & 0 & -1 & 2 \end{pmatrix}$$

- Dim: $N \times N$
- How many non-zeros?

  Answer: $3N$
- How many zeros?

  Answer: $N^2 - 3N$
- Sparsity?

  Answer: $\frac{N^2-3N}{N^2} = 1 - \frac{3}{N}$

  goes to zero as $N \to \infty$!

  Nice and sparse!

- We will be mostly concerned w/ matrices of this kind, but you should know that there are methods dealing w/ dense matrices.
- Sparse algorithms (those based on sparse matrices) tend to scale better w/ the size of the system.

Dense vs. Sparse

Direct vs. Iterative

What kind of matrix?

The next thing we need to decide concerns methods:

- Direct: Require having the whole matrix (non-zeros + locations) in memory.
- Iterative: only require knowing how to apply the matrix to a given vector.

Are out of the question for large problems (like 3+1 d problems)

- Iterative methods tend to be faster than direct ones since they scale better w/ dimension. However, they are also more sensitive to the condition number.