DQMC Simulation of the Hubbard Model

Joshua Horowitz

Fall 2011
Overview

1. The Physics
   - quadratic Hamiltonians, Hubbard model, Hubbard-Stratonovich transformation

2. The Simulation
   - Markov chain Monte Carlo

3. The System
   - Python, NumPy, cProfile, weave, EC2, StarCluster
Hopping Hamiltonian

- Lattice of single-electron states, hopping terms connecting them.

\[ \hat{H} = -t \sum_{\langle ij \rangle} a_i^\dagger a_j \]
Hopping Hamiltonian — Quadratic in Operators

- Adding chemical potential & spin multiplicity,

\[ \hat{H} = -t \sum_{\langle ij \rangle} a_{i\sigma}^\dagger a_{j\sigma} - \mu_{i\sigma} \sum_i a_{i\sigma}^\dagger a_{i\sigma} \]

\[ = \left( \begin{array} {cccc} a_{1\sigma}^\dagger & a_{2\sigma}^\dagger & \cdots \end{array} \right) \left( \begin{array} {ccc} h_{11} & h_{12} & \cdots \ h_{21} & h_{22} & \cdots \ \cdots & \cdots & \cdots \end{array} \right) \left( \begin{array} {c} a_{1\sigma} \\ a_{2\sigma} \\ \vdots \end{array} \right). \]

- Quadratic \( \Rightarrow \) Exactly solvable!

\[ Z = \text{tr} \left[ e^{-\beta \hat{H}} \right] = \det \left[ I + e^{-\beta \hat{h}} \right]. \]

Other rules for Green’s functions \( \langle a_{i\sigma}^\dagger a_{j\sigma} \rangle = \frac{\text{tr} \left[ a_{i\sigma}^\dagger a_{j\sigma} e^{-\beta \hat{H}} \right]}{\text{tr} \left[ e^{-\beta \hat{H}} \right]} \).
Adding Interactions

- Not too surprising that system is solvable $\leftrightarrow$ no interactions.
- Simplest possible interaction is on-site repulsion (fixed cost to doubly-occupied site):

$$\hat{H}_{\text{int}} = U \sum_i n_{i\uparrow} n_{i\downarrow}$$

... and here's the problem...

$$= U \sum_i a_{i\uparrow}^\dagger a_{i\uparrow} a_{i\downarrow}^\dagger a_{i\downarrow}. $$

- No longer have a quadratic Hamiltonian.
Adding Interactions – Discrete H-S Transformation

For convenience:

\[ \hat{H}_{\text{int}} : \quad U \sum_i n_{i\uparrow} n_{i\downarrow} \rightarrow U \sum_i \left( n_{i\uparrow} - \frac{1}{2} \right) \left( n_{i\downarrow} - \frac{1}{2} \right) \]

Here’s a trick:

\[ e^{-\beta U(n_{i\uparrow} - \frac{1}{2})(n_{i\downarrow} - \frac{1}{2})} = C \sum_{s_i} e^{-\beta(\lambda s_i n_{i\uparrow} - \lambda s_i n_{i\downarrow})} \]

\[ e^{-\beta \hat{H}_{\text{int}}} = C^N \sum_{\{s_i\}} e^{-\beta \sum_i (\lambda s_i n_{i\uparrow} - \lambda s_i n_{i\downarrow})} \]

\[ Z = \text{tr} \left[ e^{-\beta \hat{H}_{\text{int}}} \right] = C^N \sum_{\{s_i\}} \text{tr} \left[ e^{-\beta \sum_i (\lambda s_i n_{i\uparrow} - \lambda s_i n_{i\downarrow})} \right] \]

Sum quadratic-Hamiltonian \( Z \)'s over values of external field!
Adding Interactions – How does H-S really work?

- ↑ electrons and ↓ electrons are only connected through “randomly fluctuating” field... how does this create a repulsive interaction?
- Hubbard-Stratonovich transformation is closely related to mean-field methods.
Adding Interactions – Trotter Product

“Not so fast...”

\[ e^{-\beta(\hat{H}_0 + \hat{H}_{\text{int}})} \neq e^{-\beta \hat{H}_0} e^{-\beta \hat{H}_{\text{int}}}. \]

Happy resolution – the Trotter product formula!

\[ e^{-\beta(\hat{H}_0 + \hat{H}_{\text{int}})} = \lim_{\Delta \rightarrow 0} \left( e^{-\Delta \hat{H}_0} e^{-\Delta \hat{H}_{\text{int}}} \right)^{\beta/\Delta}. \]

(Our trace-taking techniques survive the breakup.)

H-S variables \( s_i \) are now \( s_{i,\tau} \); a field in space(i)time.
Adding Interactions – Making Measurements

- Summing over all possible \( \{s_{i,\tau}\} \) can give any measurement. For instance, the partition function:

\[
Z \approx \text{tr} \left[ \left( e^{-\Delta \tau \hat{H}_0} e^{-\Delta \tau \hat{H}_{\text{int}}} \right)^{\beta/\Delta \tau} \right] = \sum_{\{s_{i,1}\}} \text{tr} \left[ e^{-\Delta \tau \hat{H}_0} e^{-\Delta \tau \hat{H}_{\text{HS}}(\{s_{i,L}\})} \ldots e^{-\Delta \tau \hat{H}_0} e^{-\Delta \tau \hat{H}_{\text{HS}}(\{s_{i,1}\})} \right] = \sum_{\{s_{i,1}\}} \text{det} \left[ I + e^{-\Delta \tau h_0} e^{-\Delta \tau h_{\text{HS}}(\{s_{i,L}\})} \ldots e^{-\Delta \tau h_0} e^{-\Delta \tau h_{\text{HS}}(\{s_{i,1}\})} \right] .
\]

- Or correlation functions:

\[
\left\langle a_{i,\sigma}^\dagger a_{j,\sigma} \right\rangle_{\{s_{i,1}\}} = (I - M(\{s_{i,1}\}^{-1}))_{ij}
\]

\[
\left\langle a_{i,\sigma}^\dagger a_{j,\sigma} \right\rangle = \sum_{\{s_{i,1}\}} \frac{\text{det} \left[ M(\{s_{i,1}\}) \right]}{Z(p(\{s_{i,1}\}))} (I - M(\{s_{i,1}\}^{-1}))_{ij}.
\]
Markov Chain Monte Carlo – Why?

- But $\sum_{\{s_i, l\}}$ is only a conceptual possibility
  - $6 \times 6$ grid, with a single time-step? 68,719,476,736 terms
  - (Each term involves a large matrix $\times$, $(\cdot)^{-1}$, det $[\cdot]$)

- Random sampling?
  - Spend too much time in unlikely places:

![Histogram](image.png)

- (And term computation is still slow)

- Solution: Markov chain Monte Carlo
Markov Chain Monte Carlo

- General idea: Construct a Markov chain with a desired probability distribution as its stationary distribution.
  - Our case – $\{s_i, l\}$ will take a random walk, by flipping spins with appropriately chosen rates
  - It will then sample the full distribution accurately
  - Only requires knowing ratios of probabilities: very easy!

- Efficient execution comes from keeping track of Green’s-function matrices during random walk
  - test potential flips VERY quickly
  - update Green’s functions in response to a spin-flip using the Sherman-Morrison formula
  - “wrap” Green’s functions from each $(i)$time-step to the next
Results

Shown: My results for $U = 4$ and $U = 8$; vs. Scalettar
(6 × 6 grid at half-filling)
The Python Environment

- **Python**: Modern language with expressive syntax, object-oriented/functional features, and ample libraries.
- **NumPy**: Adds fast numerical arrays to Python.
- **SciPy**: Large library of math algorithms based on NumPy.

- Why? I like Python. Wanted to see if it was up to the task.
“Premature optimization is the root of all evil.” – Donald Knuth
Speeding Up Code: weave

- Put C++ code in Python:

```python
def update_G(self, G, new_G, i, Delta_coeff, det_ratio):
    coeff = float(Delta_coeff/det_ratio)
    N = self.N
    code = ""
    for (int r = 0; r < N; r++) {
        for (int c = 0; c < N; c++) {
            new_G(r,c) = G(r,c) + G(r, i)*(G(i, c) - (c==i?1:0))*coeff;
        }
    }
    ""
    weave.inline(code, ['G', 'N', 'i', 'coeff', 'new_G'],
                  type_converters=weave.converters.blitz,
                  compiler='gcc')
```

- Speed up bottlenecks without losing clarity/usability/development-speed of Python.
Commodity Clusters: Amazon EC2 & StarCluster

- Amazon EC2 offers virtual machines.
  - $0.08/core-hour, on-demand
- StarCluster configures everything.
  - shared filesystem (NFS), queueing system (SGE), etc.

- Almost certainly not competitive with academic resources\(^1\), but a fascinating resource.
  - Note: SDSC charges $0.15/core-hour, with a $2500 minimum.

- In 5min computation, with 64 cores...
Shown: My results for $U = 4$ and $U = 8$; vs. Scalettar
(6 × 6 grid at half-filling)
Problems & Future Directions

- Low temperature! Instabilities arise everywhere.
  - Dealt with so far: adaptive recomputation of Green’s functions
  - Many more to go (fancy multiplication)

- Scaling
  - Can the system handle larger lattices? More complex models?

- Physics
  - Extend system to explore the universe.