

Self-learning and Floating- block EC

Avik Sarkar and Dean Lee

Outline

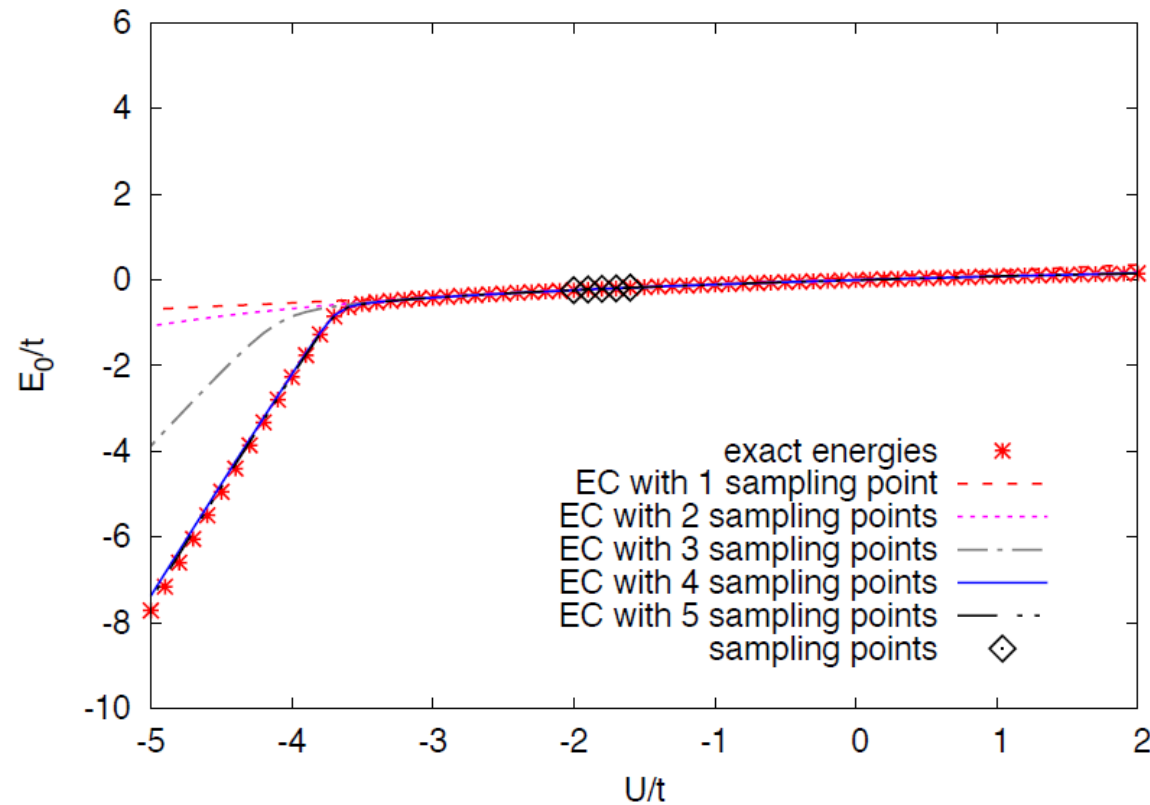
- Introduction
- Self-learning emulators
- Floating-block EC
- Summary

Problem

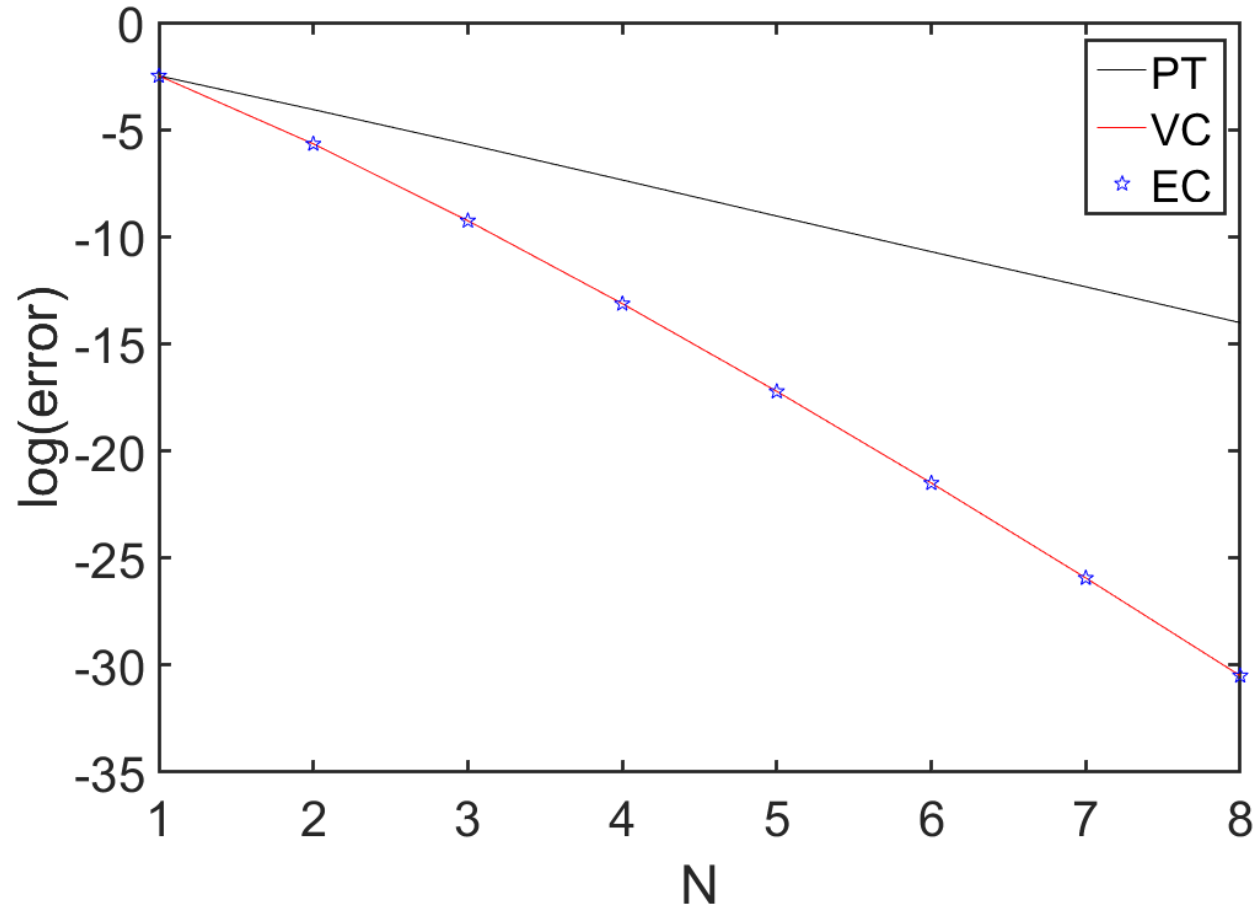
- A good emulator should be detailed enough to capture salient aspects of the mathematical problem at hand, but a highly detailed model can make it computationally expensive, hindering its usability.
- An emulator is designed using some training points where the exact data is known.
- **How many training points do we need to reach a desired accuracy?**
- **Where should we select these training points?**

Background

- EC approximates the target eigenvector with the best linear combination of training eigenvectors that minimizes error in energy.



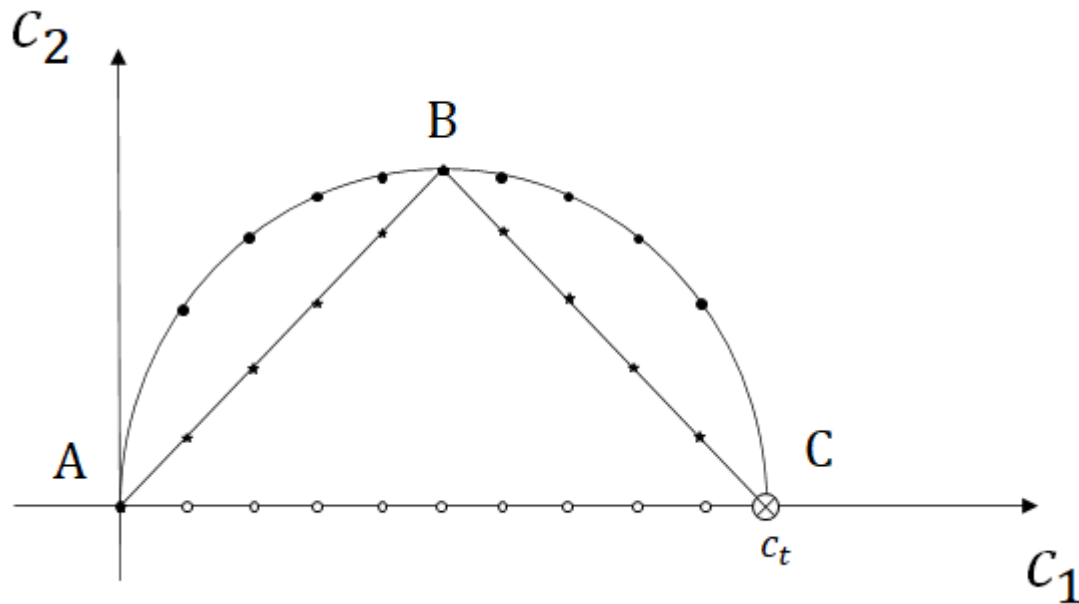
Convergence properties of EC



$$|\widetilde{v(c_t)}\rangle_{PT} = \sum_{n=0}^N |v^{(n)}(0)\rangle \frac{c_t^n}{n!}$$

Background

- Convergence determined by the orthogonality of the training eigenvectors



$$|\widetilde{v}(c_t)\rangle_{PT} = \sum_{n=0}^N |v^{(n)}(0)\rangle \frac{c_t^n}{n!}$$

Rephrasing the problem

- Input variable : $x \in D$ Output variable : y
- Emulated approximation: $f(x_i) = \tilde{y}_i$
- Assume that we can find any y_i for any x_i , but it is computationally expensive
- We want to choose a set of input points $\{x_i\}$ which we will use to build an emulator.
- Which points $\{x_i\}$ should we choose and how many?
- There is an immense amount of data, and we wish to select a subset of data points that is most useful for us = **Active Learning**.

Active-learning

- Active learning (also called “query learning,” or sometimes “optimal experimental design” in the statistics literature) is a subfield of machine learning.
- The key hypothesis is that, if the learning algorithm is allowed to choose the data from which it learns it will perform better with less training.
- It has been used in a variety of fields [1 - 4] and is often combined with other machine learning methods like neural networks.

[1] - D. Cohn, L. Atlas, and R. Ladner, Machine learning 15, 201(1994).

[2] - D. A. Cohn, Z. Ghahramani, and M. I. Jordan, Journal of artificial intelligence research 4, 129 (1996).

[3] - B. Settles, Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2009).

[4] - A. G. Ellis, R. Iskandar, C. H. Schmid, J. B. Wong, and T. A. Trikalinos, Stat Med.39, 3521 (2020), 1812.07673.

Criteria for data selection

- Iterative algorithm
- Choose the point where the emulator has largest error
- Problem: Error = $y_i - \tilde{y}_i$ requires knowledge of y_i which is hard to compute
- Have the emulator give us $f(x_i) = \tilde{y}_i + \Delta\tilde{y}_i$
- Choose the point where $\Delta\tilde{y}_i$ is largest [1, 2]
- Is there some intuitive way to find out when our emulator approximation is bad?

[1] - D. J. C. MacKay, Neural Computation 4, 590 (1992), ISSN0899-7667.

[2] - A. G. Ellis, R. Iskandar, C. H. Schmid, J. B. Wong, and T. A. Trikalinos, Stat Med.39, 3521 (2020), 1812.07673.

Emulator for solving constraint problems

- Suppose that we are emulating the solution to a set of constraint equations $G_i(y, c) = 0$.
- Examples: Finding the roots of a polynomial with variable coefficients, solution of a differential equation, eigenvector of a large matrix, etc.
- Constraint equations lets us test how good our emulated approximation is.
 - $G(\tilde{y}_i, c) = 0.001 \sim \tilde{y}_i$ is probably good solution
 - $G(\tilde{y}_j, c) = 1000 \sim \tilde{y}_j$ is probably bad solution

Self-learning Emulator

- Data selection criteria for self-learning emulator:
 - Use the constraint equations to find an estimate of the exact error.
 - Each iteration we emulate and find the error estimate everywhere
 - Find the point where the error estimate is largest and take that as the next training point.
- The computational advantage is the acceleration factor of the emulator itself.
- At the end find the exact error at a few points and use that to estimate the exact error.

Self-learning Cubic Spline Interpolator

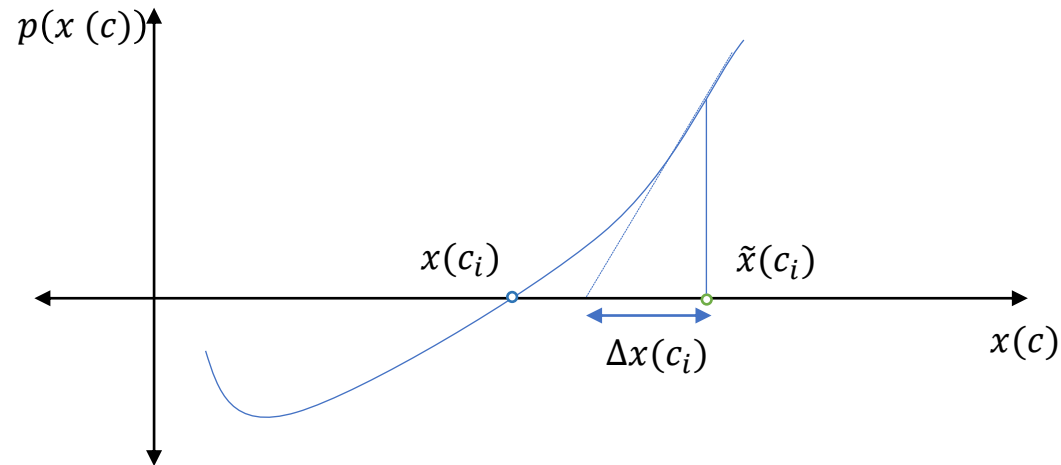
- $p(x) = 5x^5 + cx^4 + 3x^3 + 2x^2 + 1x + 1$
- Let $x(c)$ be the smallest real root.
- From few data points $\{c_i\}$, we interpolate to all domain $c \in D$ using natural cubic spline interpolation.
- Emulator approximation: $f(c_i) = \tilde{x}(c_i)$
- Exact error: $\Delta x(c_i) = x(c_i) - \tilde{x}(c_i)$
- Constraint equation: $p(x(c_i)) = 0$
- We want an error estimate $F[p(\tilde{x}(c)), c]$ which is proportional to the exact error $\Delta x(c)$ so that they have maxima at the same point.

$$\log |\Delta x(c)| = \log F[p(\tilde{x}(c)), c] + A + B(c)$$

Error Estimate function

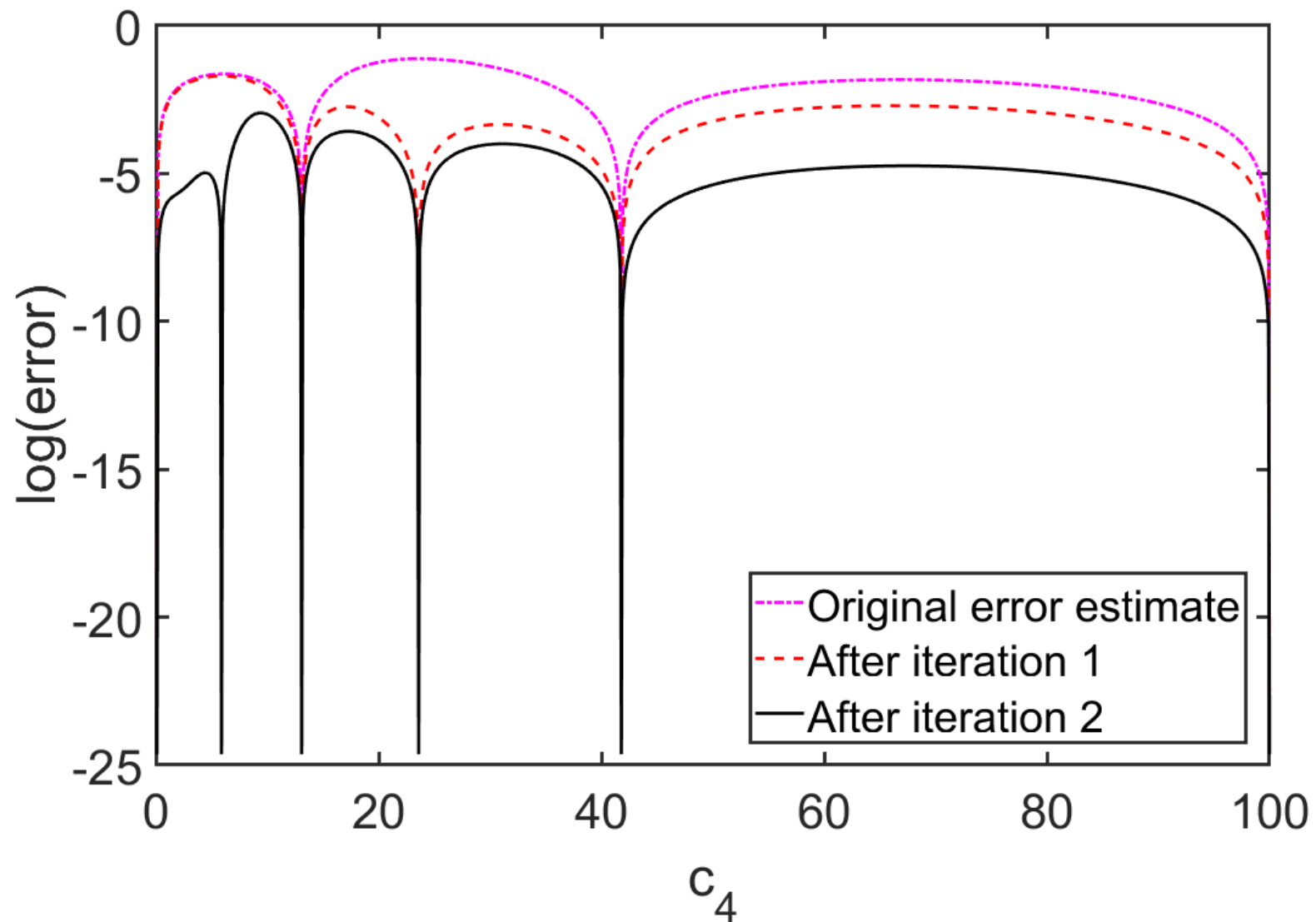
- We can estimate the error using the Newton-Raphson method,

$$\Delta x(c) \approx \frac{|p(\tilde{x}(c))|}{\sqrt{|p'(\tilde{x}(c))|^2 + \epsilon^2}}$$

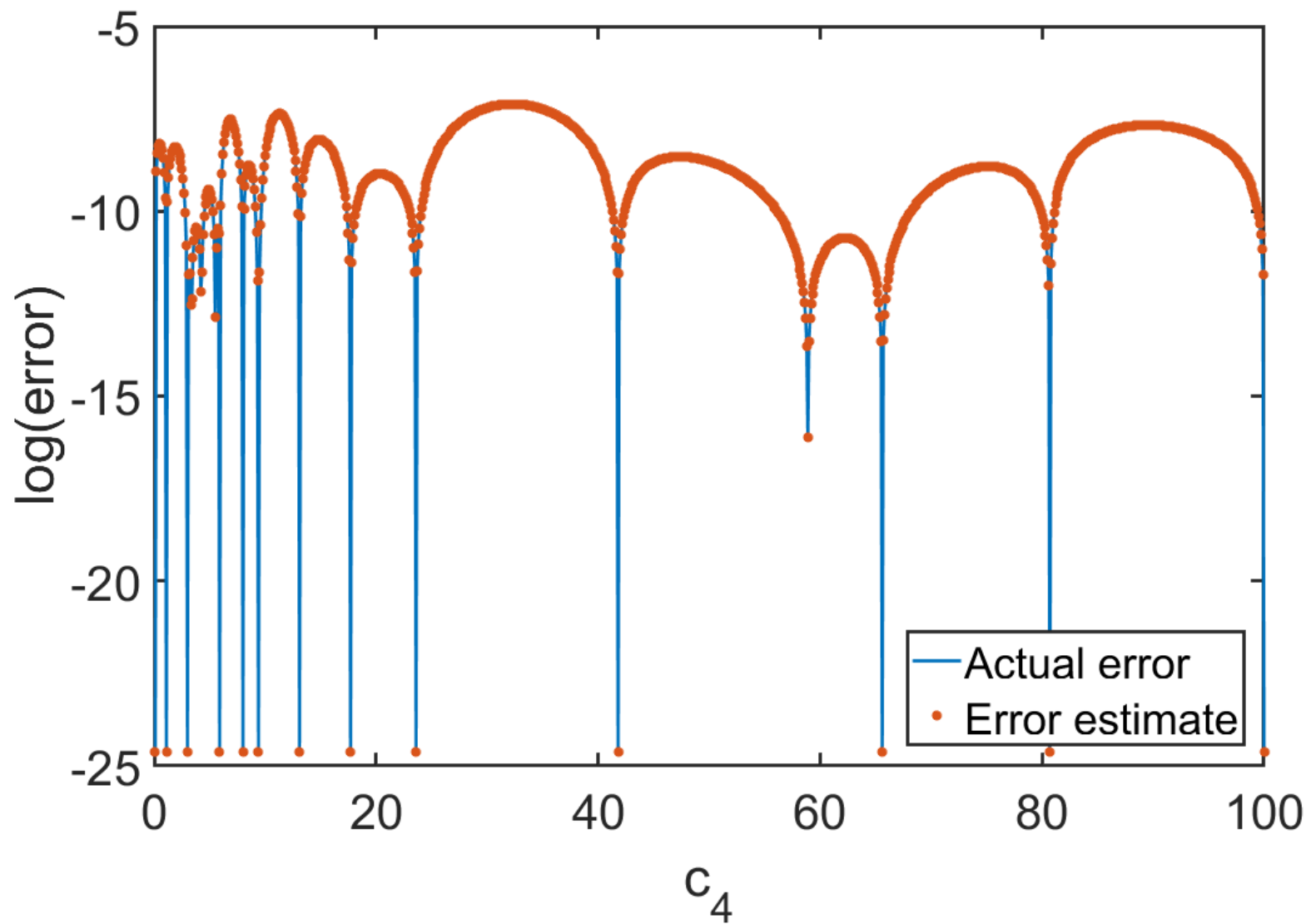


- As $\Delta x(c) \rightarrow 0$, this estimate becomes the actual error, and thus we take the above expression to be our error estimate.

Self-learning process



Actual Error



EC Emulator

- Assume that we can find the exact solution at some training points $c_i = c_{i_0}, \dots, c_{i_N}$. Denote them by $\{|\psi(c_{i_0})\rangle, |\psi(c_{i_1})\rangle, \dots, |\psi(c_{i_N})\rangle\}$.
- Effectively, eigenvector continuation attempts to approximate $|\psi(c_t)\rangle$ by finding the best linear combination of $|\psi(c_{i_0})\rangle, \dots, |\psi(c_{i_N})\rangle$, minimizing error in energy.
- We want to optimize our selection of training points $c_i = c_{i_0}, \dots, c_{i_N}$.

Self-learning EC

- Algorithm to find best training points for EC.
- Constraint Equation: $H(c)|\psi(c)\rangle = E(c)|\psi(c)\rangle$
- EC approximation: $\tilde{E}(c), \tilde{\psi}(c)$
- Error: $\Delta|\psi(c)\rangle = \|\ |\psi(c)\rangle - |\tilde{\psi}(c)\rangle\|$

- Error estimate:
$$F(c) = \sqrt{\frac{\langle \tilde{\psi}(c) | [H(c) - \tilde{E}(c)]^2 | \tilde{\psi}(c) \rangle}{\langle \tilde{\psi}(c) | [H(c)]^2 | \tilde{\psi}(c) \rangle}}$$

Four particles in lattice

- Consider 4 distinguishable particles in a 3D lattice with zero-range interaction.
- Lattice volume: $L = 4^3$, Hamiltonian dimension: 262,144.
- Generalization of Bose-Hubbard model with 4 bosons [1] or 4 two-component fermions on lattice [2 - 4].
- $H = H_{Free} + \sum_{i<j} \sum_n c_{ij} \rho_i(n) \rho_j(n)$; $\rho_i(n)$ = density operator
- Learning the 6-dimensional subspace with matrix size 262,144 is a challenge, but is possible with self-learning EC.

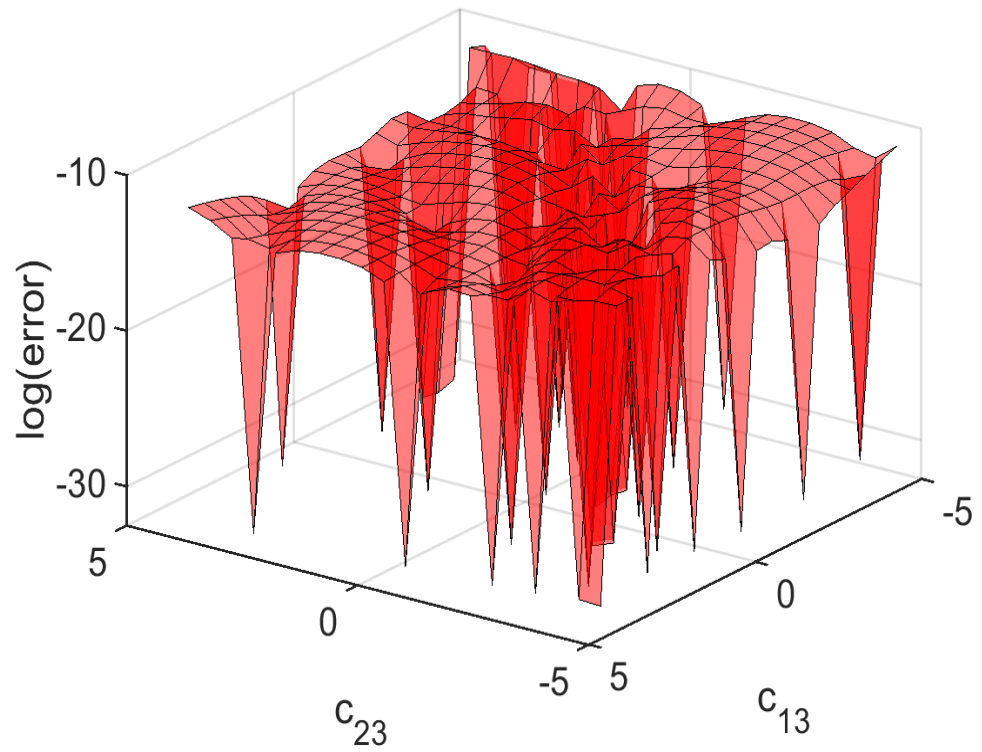
[1] - D. Frame, R. He, I. Ipsen, D. Lee, D. Lee, and E. Rrapaj, Phys.Rev. Lett.121, 032501 (2018), 1711.07090.

[2] - A. Sarkar and D. Lee, Phys. Rev. Lett.126, 032501 (2021).

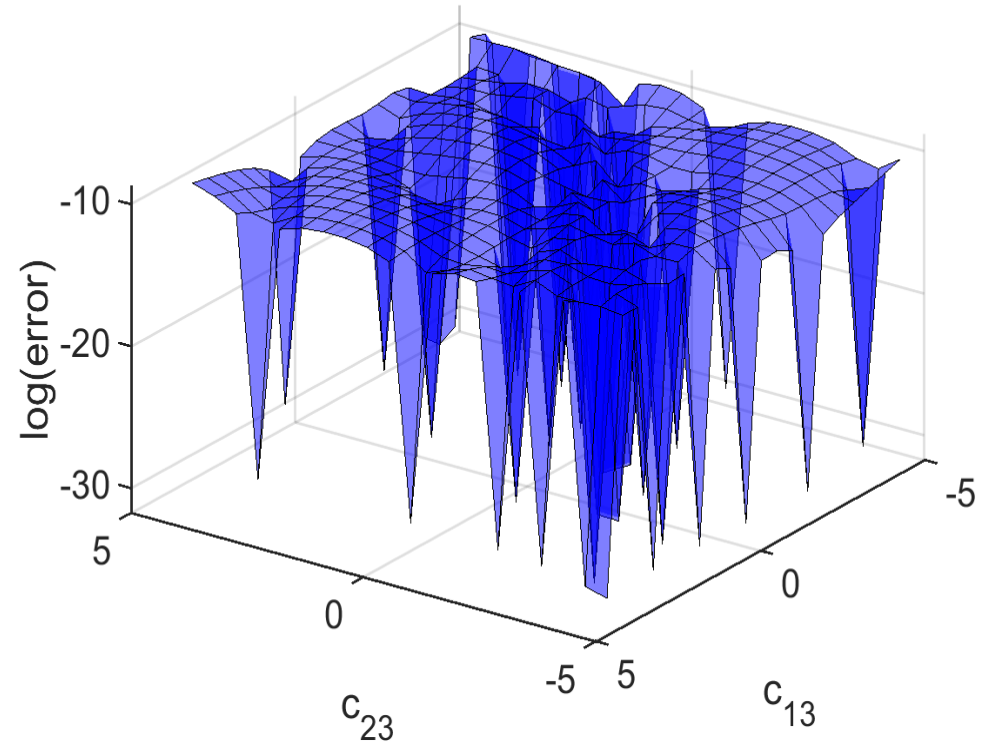
[3] – S. Bour, Xin Li, D. Lee, Ulf-G. Meißner, L. Mitas (2011), 1104.2102

[4] – S.Elhatisari, K. Katterjohn, D. Lee, Ulf-G. Meißner, G. Rupak (2017), 1610.09095

Two parameter variation

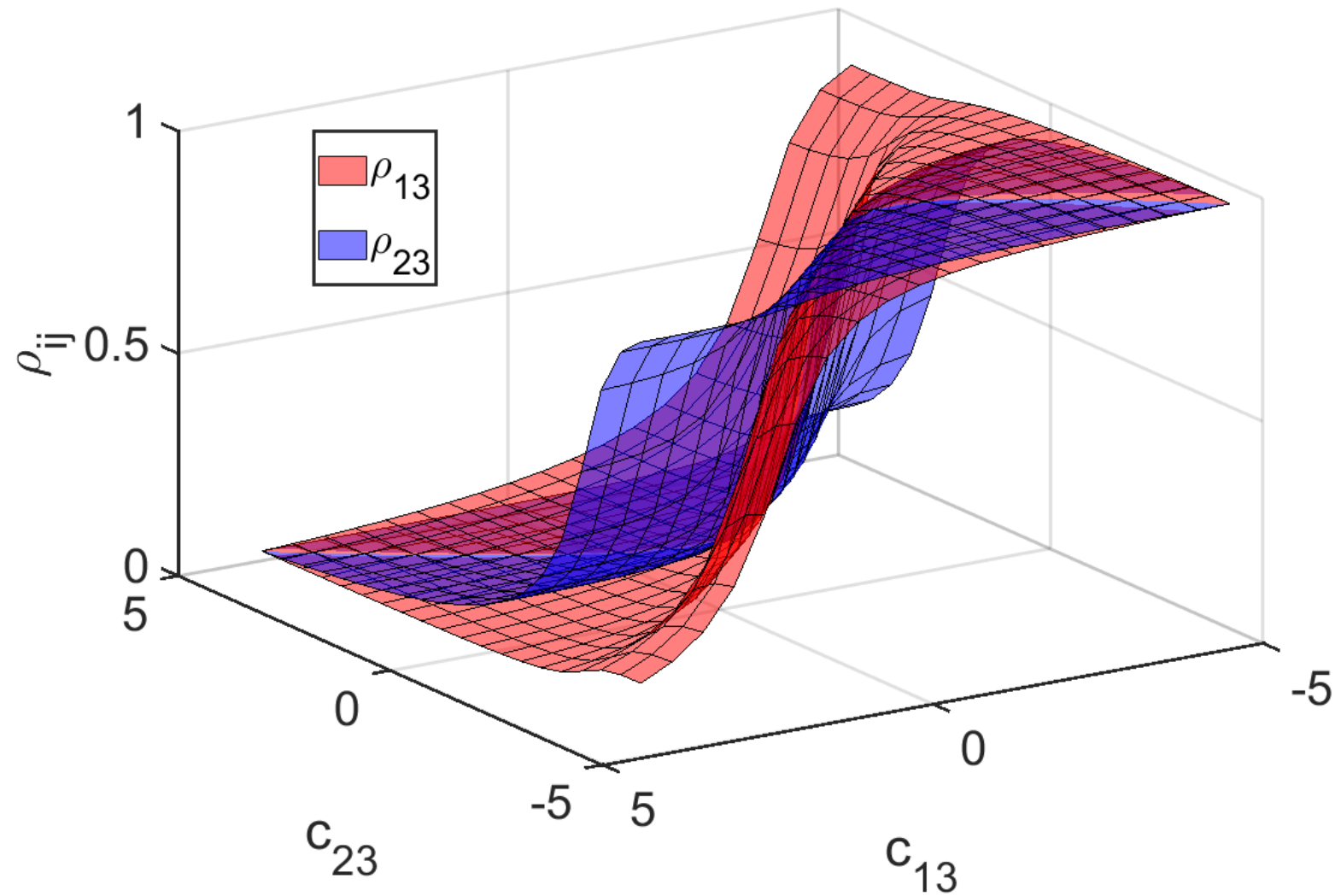


Actual Error



Error Estimate

Short range correlations



Summary

- Self-learning emulator – a new algorithm to train emulators.
- We use the emulator directly for training, and the computational advantage is the acceleration factor of the emulator itself, which can grow to five orders of magnitude or more for large systems. [1]
- It can be used with any emulator for solving a system of constraint equations – solving a system of algebraic equations, linear and nonlinear differential equations, and linear and nonlinear eigenvalue problems.
- We used it to find training points for EC in a six-dimensional vector space of eigenvectors. Using EC, we emulated over 10^6 points and had error about 10^{-3} with 80 training points.

Floating-block EC

- Applying EC to quantum Monte Carlo simulations using Lattice EFT.
- Key idea: Euclidean time evolution with transfer matrix formalism

$$\frac{\langle \psi_{initial} | e^{-H(c)\delta t} | \dots | H(c) | \dots | e^{-H(c)\delta t} | \psi_{initial} \rangle}{\langle \psi_{initial} | e^{-H(c)\delta t} | \dots | e^{-H(c)\delta t} | \psi_{initial} \rangle}$$

- Notation:
$$H_{ij}(c_t) = \frac{\langle \psi_{init} | c_i | \dots | c_i | H(c_t) | c_j | \dots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_i | \dots | c_i | c_j | \dots | c_j | \psi_{init} \rangle}$$

- Problem:
$$N_{ij} = ??$$

Previous work

$$H_{ij}(c_t) = \frac{\langle \psi_{init} | c_i | \cdots | c_i | H(c_t) | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_m | \cdots | c_m | c_n | \cdots | c_n | \psi_{init} \rangle}$$

$$N_{ij}(c_t) = \frac{\langle \psi_{init} | c_i | \cdots | c_i | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_m | \cdots | c_m | c_n | \cdots | c_n | \psi_{init} \rangle}$$

- Norm matrix ill-conditioned

Floating-blocks

$$\frac{\langle \psi_{init} | c_i | \cdots | c_i | c_j | \cdots | c_j | c_i | \cdots | c_i | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_i | \cdots | c_i | c_i | \cdots | c_i | c_j | \cdots | c_j | c_j | \cdots | c_j | \psi_{init} \rangle}$$

$$\frac{\langle \psi_{init} | c_i | \cdots | c_i | \sum_{k_1} |\psi_{k_1}\rangle \langle \psi_{k_1} | c_j | \cdots | c_j | \sum_{k_2} |\psi_{k_2}\rangle \langle \psi_{k_2} | c_i | \cdots | c_i | \sum_{k_3} |\psi_{k_3}\rangle \langle \psi_{k_3} | c_j | \cdots | c_j | \sum_{k_4} |\psi_{k_4}\rangle \langle \psi_{k_4} | \psi_{init} \rangle}{\langle \psi_{init} | c_i | \cdots | c_i | \sum_{k_5} |\psi_{k_5}\rangle \langle \psi_{k_5} | c_j | \cdots | c_j | \psi_{init} \rangle}$$

$$= \frac{\langle \psi_{init} | \psi(c_i) \rangle \langle \psi(c_i) | \psi(c_j) \rangle \langle \psi(c_j) | \psi(c_i) \rangle \langle \psi(c_i) | \psi(c_j) \rangle \langle \psi(c_j) | \psi_{init} \rangle}{\langle \psi_{init} | \psi(c_i) \rangle \langle \psi(c_i) | \psi(c_j) \rangle \langle \psi(c_j) | \psi_{init} \rangle}$$

$$\frac{\langle \psi_{init} | c_i | \cdots | c_i | c_j | \cdots | c_j | c_i | \cdots | c_i | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_i | \cdots | c_i | c_i | \cdots | c_i | c_j | \cdots | c_j | c_j | \cdots | c_j | \psi_{init} \rangle} = |\langle \psi(c_i) | \psi(c_j) \rangle|^2$$

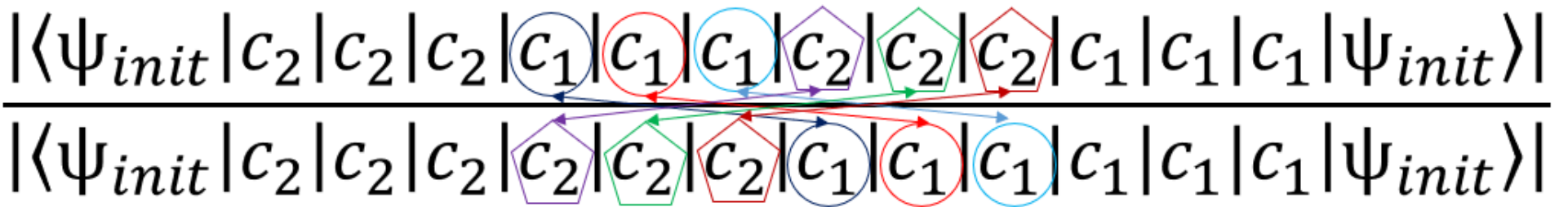
Floating-block EC

$$H_{ij}(c_t) = \frac{\langle \psi_{init} | c_i | \cdots | c_i | H(c_t) | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_m | \cdots | c_m | c_n | \cdots | c_n | \psi_{init} \rangle}$$

$$N_{ij} = \left(\frac{\langle \psi_{init} | c_i | \cdots | c_i | c_j | \cdots | c_j | c_i | \cdots | c_i | c_j | \cdots | c_j | \psi_{init} \rangle}{\langle \psi_{init} | c_i | \cdots | c_i | c_i | \cdots | c_i | c_j | \cdots | c_j | c_j | \cdots | c_j | \psi_{init} \rangle} \right)^{\frac{1}{2}}$$

- Extrapolate? – restrained by error/need more computation time
- Fast emulators
- Two computational tricks

Shifting Auxiliary Fields

$$N^2 = \frac{|\langle \Psi_{init} | c_2 | c_2 | c_2 | \underbrace{c_1}_{\text{blue circle}} | \underbrace{c_1}_{\text{red circle}} | \underbrace{c_1}_{\text{blue circle}} | \underbrace{c_2}_{\text{purple pentagon}} | \underbrace{c_2}_{\text{green pentagon}} | \underbrace{c_2}_{\text{red pentagon}} | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}{|\langle \Psi_{init} | c_2 | c_2 | c_2 | \underbrace{c_2}_{\text{purple pentagon}} | \underbrace{c_2}_{\text{green pentagon}} | \underbrace{c_2}_{\text{red pentagon}} | \underbrace{c_1}_{\text{blue circle}} | \underbrace{c_1}_{\text{red circle}} | \underbrace{c_1}_{\text{blue circle}} | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}$$


3-step Calculation

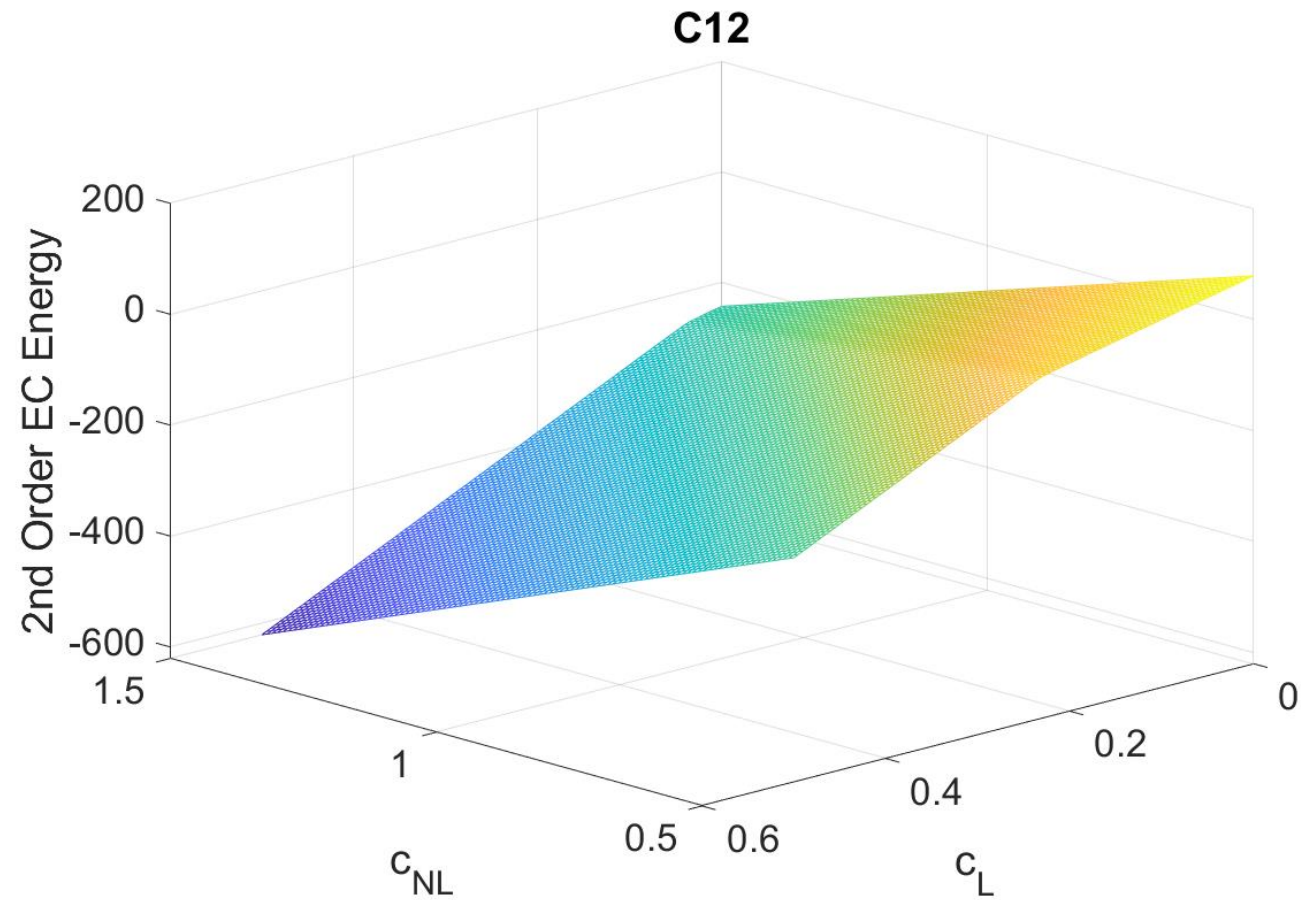
$$\begin{aligned}
 N_1 &= \frac{|\langle \Psi_{init} | c_2 | c_2 | c_2 | \textcircled{c_1} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | c_1 | c_1 | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}{|\langle \Psi_{init} | c_2 | c_2 | c_2 | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_1} | c_1 | c_1 | c_1 | c_1 | c_1 | \Psi_{init} \rangle|} \\
 N_2 &= \frac{|\langle \Psi_{init} | c_2 | c_2 | c_2 | c_1 | \textcircled{c_1} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | c_1 | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}{|\langle \Psi_{init} | c_2 | c_2 | c_2 | c_1 | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_1} | c_1 | c_1 | c_1 | c_1 | \Psi_{init} \rangle|} \\
 N_3 &= \frac{|\langle \Psi_{init} | c_2 | c_2 | c_2 | c_1 | c_1 | \textcircled{c_1} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}{|\langle \Psi_{init} | c_2 | c_2 | c_2 | c_1 | c_1 | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_2} | \textcircled{c_1} | c_1 | c_1 | c_1 | \Psi_{init} \rangle|}
 \end{aligned}$$

Nuclear binding near a quantum phase transition

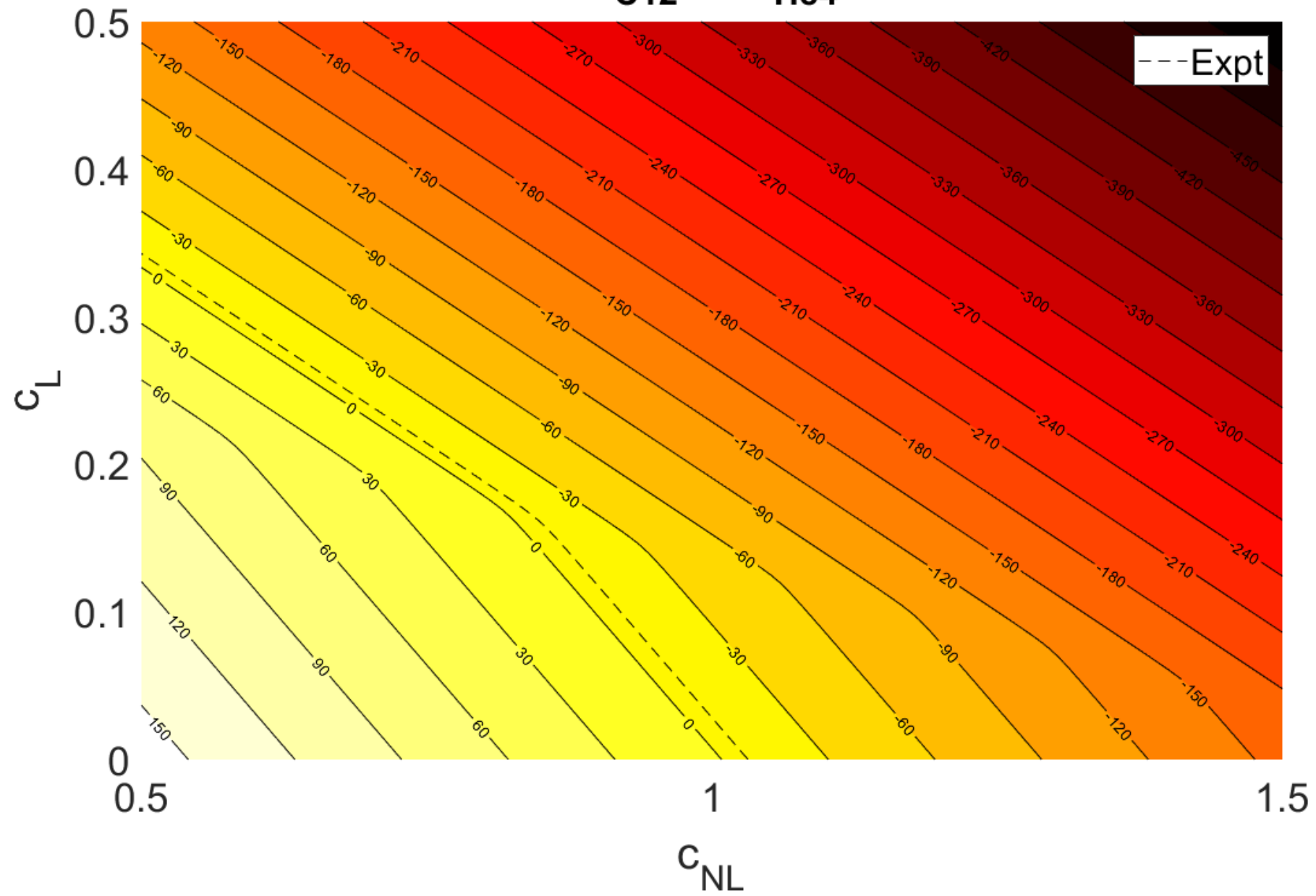
Serdar Elhatisari,¹ Ning Li,² Alexander Rokash,³ Jose Manuel Alarcón,¹ Dechuan Du,² Nico Klein,¹ Bing-nan Lu,² Ulf-G. Meißner,^{1,2,4} Evgeny Epelbaum,³ Hermann Krebs,³ Timo A. Lähde,² Dean Lee,⁵ and Gautam Rupak⁶

- Study phase transition with two different interactions
- $H_a = K + V_L$, $H_b = K + V_{NL}$
- Tune smearing parameter σ such that both interactions reproduce ${}^4\text{He}$ energy correctly.
- For larger systems (${}^8\text{Be}$, ${}^{12}\text{C}$, ${}^{16}\text{O}$), V_L interaction produces overbinding, and V_{NL} interaction makes the system alpha gas.
- EC emulate: $H(c_L, c_{NL}) = K + c_L V_L + c_{NL} V_{NL}$

EC emulator



$$E_{C12} - 3E_{He4}$$



12C EC error

(c_L, c_{NL})	Energy	2 nd order EC	3 rd order EC	4 th order EC
(0.8,0.2)	-338.57	-330.63	-333.15	-333.24
(0.3,0.7)	-141.11	-139.63	-141.09	-141.1
(0.4,0.7)	-217.73	-217.70	-217.71	-217.72
(0.4,0.8)	-259.35	-258.19	-258.31	-258.32
(0.8,0.3)	-382.41	-371.08	-374.24	-374.34
(0.8,0.8)	-606.63	-574.23	-579.97	-580.12
(0.2,0.6)	-41.54	-31.91	-37.75	-41.64
(0.2,0.2)	31.26	101.48	99.18	69.72

- Training points: (0.5,0.5), (0.2,0.8), (0.0,1.0), (0.2,0.6)
- Error in each EC estimate can be calculated using trimmed sampling

Summary

- New algorithm to perform EC in lattice Monte Carlo.
- Error limits what can be done, but extrapolation is possible
- Can be used to build EC emulators

**Thank you for
your attention**