SRG applied to Quantum Dots - what I have done so far

Sarah Reimann

Department of Physics, University of Oslo, Norway

November 13, 2012

Aim of the thesis

- Study the ground state of closed-shell systems of quantum dots in two dimensions
- Method: Similarity renormalization group (SRG) method
- Use of the same methodology discussed in the paper 'Similarity renormalization group for nucleon-nucleon interactions' by S.K.Bogner et al.

Implemented equations

The Hamiltonian:

$$H = T_{\text{rel}} + V$$

T_{rel}: relative kinetic energy, V: interaction part

Flow of the Hamiltonian:

$$\frac{dH_s}{ds} = [\eta_s, H_s]$$

Choice of generator:

$$\eta_s = [T_{\text{rel}}, H_s] = [T_{\text{rel}}, V_s]$$

Only interaction part V dependent on flow parameter s!
Chosen basis: Harmonic oscillator basis

$$T_{\text{rel}} = \begin{pmatrix} T_0 & 0 & \dots & 0 \\ 0 & T_1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & T_n \end{pmatrix}$$

with $T_i = \sum_{i=1}^{N} \epsilon_i = \sum_{i=1}^{N} \hbar\omega(2n_i + |m_i| + 1)$, N - number of particles.

Implemented equations

With this choice of T_{rel}:

$$\eta_{ij}(s) = (\epsilon_i - \epsilon_j)\, V_{ij}(s).$$

The flow of the matrix elements is then given by

$$\frac{dH_{ij}}{ds} = \frac{dV_{ij}}{ds} = -(\epsilon_i - \epsilon_j)^2 V_{ij}(s) + \sum_k (\epsilon_i + \epsilon_j - 2\epsilon_k)\, V_{ik}(s) V_{kj}(s). \tag{1}$$

General procedure

1) Specify the number of particles/shells.
2) Set up the **basis states in M-scheme**, using binary representation. Since I am interested in the ground state, I take only those states where $M = M_s = 0$ (H is block diagonal).
3) Set up of the **initial Hamiltonian matrix**, dimension $n \times n$. Interaction matrix elements are computed using the code of S. Kvaal.
4) **Solve the system** of $n(n+1)/2$ coupled first-order differential equations using the SRG method (since the Hamiltonian is symmetric, I only consider the upper triangular part)

Details - SRG method

- ODE solver: Algorithm by Lawrence Shampine, Marilyn Gordon. C++ version found on netlib.org
- Derivative function
  - Flow parameter s, related to λ by: $\lambda = s^{-1/2}$
  - In terms of λ, this means for the flow

$$\frac{dV_{ij}(s(\lambda))}{ds} = -\frac{2}{\lambda^3}\frac{dV_{ij}(\lambda)}{d\lambda}$$

  - Alltogether: Each time the derivative-function is called, compute for each interaction element $V_{ij}(s)$

$$\frac{dV_{ij}(s(\lambda))}{ds} = -\frac{2}{\lambda^3}\left\{-(\epsilon_i - \epsilon_j)^2 V_{ij}(\lambda) + \sum_k (\epsilon_i + \epsilon_j - 2\epsilon_k)\, V_{ik}(\lambda)V_{kj}(\lambda)\right\}.$$

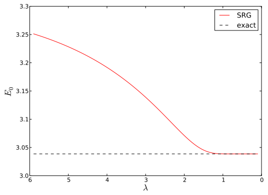
Some results for the ground state energy

	R = 3		R = 4		R = 5	
N	SRG	FCI	SRG	FCI	SRG	FCI
2	3.038604576	3.038604576	3.025230582	3.025230582	3.01760623	3.01760623
6	21.42058830	21.42058830	20.41582765	20.41582765	...	...
12	-	-	70.31250219	70.31250218	...	...

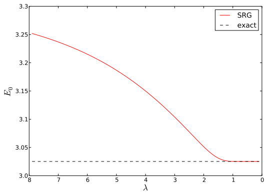
Table: E_0 in units of $\hbar\omega = 2.84$ meV. For $N = 2$ particles I performed calculations up to $R = 9$ shells and obtained always exactly the same result as with exact diagonalization.

Pro: SRG converges to the exact ground state energy for all studied systems.
Con: The required timed is exceedingly large, much larger than exact diagonalization, therefore until now just very small systems in reasonable times possible (see next slides)

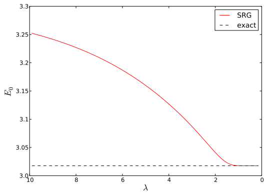
Pro: convergence of E_0!



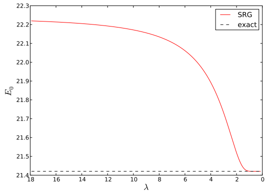
(a) $N = 2, R = 3$



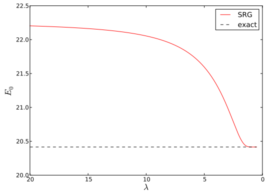
(b) $N = 2, R = 4$



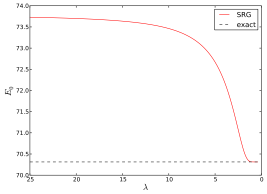
(c) $N = 2, R = 5$



(d) $N = 6, R = 3$



(e) $N = 6, R = 4$



(f) $N = 12, R = 4$

Pro: Suppression of off-diagonal matrix elements!

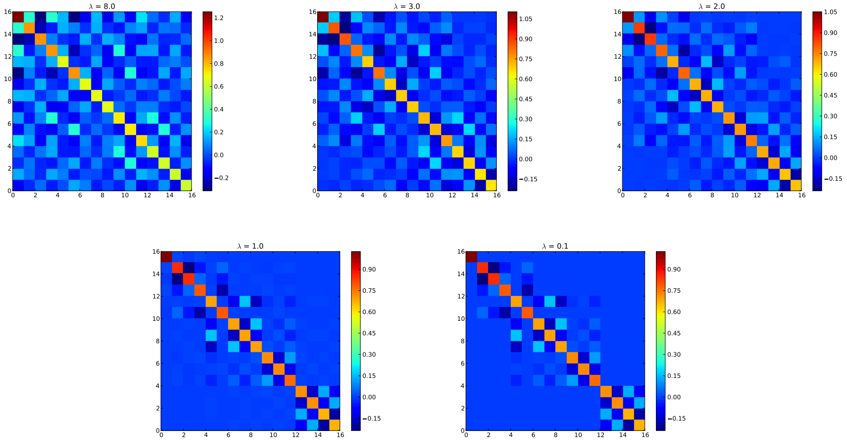


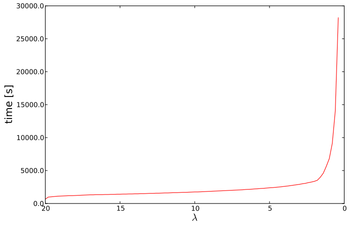
Figure: Example with $N = 2$ particles and $R = 4$ shells

Con: Time usage compared to simple exact diagonalization ...

Example: $N = 6, R = 4$, on one processor

λ	E_0	CPU time in s
20.0	**2**2.20366072	660
10.0	**2**2.05448854	1756
3.0	**2**0.99839817	2908
2.0	**20**.57836962	3363
1.4	**20.4**3153289	4608
1.0	**20.416**04434	6813
0.8	**20.41582**988	9126
0.6	**20.41582765**	14039

Table: E_0 in units of $\hbar\omega = 2.84$ meV, from $\lambda = s^{-1/2}$ follows that $[\lambda] = [E]$. The bold letters indicate correct digits.



Time exact diagonalization (standard Armadillo method): $3s$ $(E_0 = 20.41582765)$
About 90% of the time spent in derivative function, increasing with dimension of H

Problem: Fast increase of number basis states

N	$R = 3$	$R = 5$	$R = 7$
2	8	29	72
6	64	16451	594118
12	-	1630953	579968

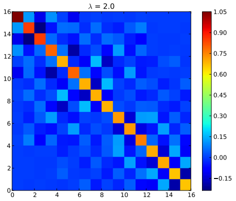
Table: Number n of relevant basis states in M-scheme with constraint $M = M_s = 0$.

Problem: Each call of the derivative function is of order $\mathcal{O}(n^3/2)$!!!
This explains the large CPU time ...

Strategy 1: Reduce gradually size of the problem

- Idea: After some integration, matrix elements far off the diagonal are **vanished** and will **stay zero**
- $\Rightarrow$ those indices i, j **not needed** any more when computing the derivatives

$$\frac{dV_{ij}}{ds} = -(\epsilon_i - \epsilon_j)^2 V_{ij}(s) + \sum_k (\epsilon_i + \epsilon_j - 2\epsilon_k)\, V_{ik}(s) V_{kj}(s)$$



Therefore:

- Save a list with all elements i, j that are needed for the derivative
- In certain intervals, delete those elements which are not needed any more (= where $V_{ij}(s)$ and $dV_{ij}(s)/ds$ are zero)

Strategy 1: Reduce gradually size of the problem

(a) $N = 2, R = 8, n = 104$

λ	10.0	5.0	1.0	0.6	0.4
skipped	0	0	0	459	1943

(b) $N = 2, R = 9, n = 145$

λ	10.0	5.0	1.0	0.8	0.6	0.4
skipped	0	0	12	393	1829	4921

(c) $N = 6, R = 4, n = 1490$, Final time: 4162s vs. 4264s

λ	20.0	10.0	1.0	0.9	0.8	0.7	0.6
skipped	7869	8036	8055	8578	11304	25072	68732

Table: Reducing the size of the problem. Second line shows number of skipped matrix elements.

Strategy 2: Usage of optimized library routines

Second attempt: Take benefit of optimized matrix-matrix multiplication routines
Flow equation

$$\frac{dV_s}{ds} = [\eta_s, H_s]$$

can be implemented as matrix-matrix multiplication with $\eta_{ij}(s) = (\epsilon_i - \epsilon_j)\, V_{ij}(s)$ and $H_{ij}(s) = \epsilon_i \delta_{ij} + V_{ij}(s)$

Usage: Blas routine for symmetric matrices $(V_{ij}(s) = V_{ji}(s))$

Result:

N	R	time [s]	time$_{\text{strategy1}}$ [s]
2	9	15.78	3.7
2	10	26.9	10.8

Table: Time comparison between both methods to reduce CPU time

Reasons for more time:

- The matrix multiplication involves **more flops** than the pre-computed expression in Eq.(1)
- **No skipping** of elements possible as in the ansatz of the previous slides

Open questions

Until now correct results, but exceedingly slow

- How to get a speedup?
- The problem lies in the $\mathcal{O}(n^3/2)$ flops each time the derivative function is called

For possibly larger systems

- The required memory to store the whole matrix explodes with number of particles/shells
- Is there a smart way not to store the whole matrix (all interaction elements for the ODE solver)?