Department of PHYSICS and ASTRONOMY

Contributions to Automated Algebra for the Few-Body Problem via Trotter-Suzuki Factorization

Kean Leung
University of North Carolina at Chapel Hill

Introduction

The goal of the project is to generalize an approach recently put forward by Prof. Joaquin Drut and PhD student Yaqi Hou, to calculate the thermodynamics of few- and many- body systems, using automated algebra approaches. Usually, analytic approaches to interacting multi-particle quantum mechanics are restricted to the few-body problem at best. Numerical approaches are essentially always used for $N > 3$ particles.

Our approach introduces imaginary time τ so that $e^{-\beta\hat{H}} = (e^{-\tau\hat{H}})^{N_\tau}$ where $e^{-\tau\hat{H}} = e^{-\tau(\hat{T}+\hat{V})}$ is approximated by Trotter-Suzuki (TS) factorization as $e^{-\frac{\tau}{2}\hat{T}}e^{-\tau\hat{V}}e^{-\frac{\tau}{2}\hat{T}}$. Using automated algebra over the discretized space-time lattice, and taking the continuum limit with decreasing τ, we aim to tackle up to $N = 7$ completely analytically. The main caveat is that increasing N results in a steep cost for each order in the approximation. Furthermore, while the approach is analytic, it does not return compact formulas. Rather, it returns very long expressions that can only be handled by a computer. The upshot is that parameters like the dimension of the problem or the coupling constant appear explicitly in the final expressions and can therefore be investigated analytically, which is a property that no other method provides.

The proposed project involves the generalization of the above approach to systems with four-body interactions, whereas the original method was only developed for two-body interactions. Systems with four-body forces are interesting for many reasons. One of them is that, by renormalization group flow, softening of repulsive-core two-body forces generates three- and four-body forces. For the purposes of this project, the motivation will be the realization of the so-called unitary limit, which is a universal regime of many-body physics in 3D, as a 1D gas with four-body interactions. Such a model was put forward by Nishida and Son [1] a few years ago and has been studied partially numerically using specialized Monte Carlo methods [2]. The objective of this project is to calculate the virial coefficients of that system, as high as possible in the virial expansion and in the coarse-lattice approximation.

My Role

My role in this project is to find ways to consolidate, simplify, and improve the performance of the codes. Summarized below are three coding contributions that I have made.

Contribution 1

Let $\mathcal{Z} = \text{tr}\left[e^{-\beta(\hat{H}-\mu\hat{N})}\right] = \sum_{N=0}^{\infty} Q_N z^N$ be the grand-canonical partition function. Through the virial expansion $-\beta\Omega = \ln\mathcal{Z} = Q_1\sum_{n=1}^{\infty} b_n z^n$, the virial coefficients b_n can be computed in terms of the N-body canonical partition function Q_N.

```
bn[n_] := (
   barray = Array[b, n];
   barray[[1]] = 1;
   series = Series[Exp[Q[1]*barray*z^Range[n]], z, 0, n];
   Z = SeriesCoefficient[Apply[Times, series], n];
   Return[Simplify[Solve[Z == Q[n], b[n]]]])
```

For two-species ($\uparrow$ and $\downarrow$) many-fermion system, $\mathcal{Z} = \sum_{M,N} Q_{MN} z_\uparrow^M z_\downarrow^N$ can be written in integral form $\int \mathcal{D}\sigma\ \det(1+z_\uparrow\mathcal{U})\ \det(1+z_\downarrow\mathcal{U})$ where $\mathcal{U}$ is a single-particle matrix that encodes the system dynamics. For two-species many-boson system, $\mathcal{Z} = \int \mathcal{D}\sigma\ \det^{-1}(1-z_\uparrow\mathcal{U})\ \det^{-1}(1-z_\downarrow\mathcal{U})$. We compute Q_{MN} in terms of path integrals $\int \mathcal{D}\sigma$ over tr U, tr U^2, tr U^3, ... as follows:

```
QmnFermion[n_, m_] := (
   k = Max[n, m];
   series = Series[Exp[(Array[U, k])
             (((-1)^(Range[k] - 1) z^(Range[k]))/Range[k])],{z, 0, k}];
   ans = Apply[Times, series];
   Return[SeriesCoefficient[ans, n] SeriesCoefficient[ans, m]];)
(* We write U[1], U[2], U[3] to denote tr U, tr U^2, tr U^3,...resp. *)

QmnBoson[n_, m_] := (
   k = Max[n, m];
   series1 = Series[Exp[(Array[U1,k])((z^(Range[k]))/Range[k])],{z,0,k}];
   series2 = Series[Exp[(Array[U2,k])((z^(Range[k]))/Range[k])],{z,0,k}];
   ans1 = Apply[Times, series1];
   ans2 = Apply[Times, series2];
   Return[SeriesCoefficient[ans1, n] SeriesCoefficient[ans2, m]];)
```

For one-species system, Q_N is computed as:

```
QnFermion[n_] := (
   series = Series[Exp[2 (Array[U, n])
            (((-1)^(Range[n] - 1) z^(Range[n]))/ Range[n])], {z, 0, n}];
   ans = Apply[Times, series];
   Return[Simplify[SeriesCoefficient[ans, n]]])

QnBoson[n_] := (
   series = Series[Exp[2(Array[U,n])((z^(Range[n]))/Range[n])],{z,0,n}];
   ans = Apply[Times, series];
   Return[Simplify[SeriesCoefficient[ans, n]]])
```

For three-species system, Q_{RST} is computed as:

```
QrstFermion[r_, s_, t_] := (
   k = Max[r, s, t];
   series = Series[Exp[(Array[U, k])
            (((-1)^(Range[k] - 1) z^(Range[k]))/Range[k])], {z,0,k}];
   ans = Apply[Times, series];
   Return[FullSimplify[SeriesCoefficient[ans,r]
         SeriesCoefficient[ans,s] SeriesCoefficient[ans,t]]];)

QrstBoson[r_, s_, t_] := (
   k = Max[r, s, t];
   series1 = Series[Exp[(Array[U1,k])((z^(Range[k]))/Range[k])],{z,0,k}];
   series2 = Series[Exp[(Array[U2,k])((z^(Range[k]))/Range[k])],{z,0,k}];
   series3 = Series[Exp[(Array[U3,k])((z^(Range[k]))/Range[k])],{z,0,k}];
   ans1 = Apply[Times, series1];
   ans2 = Apply[Times, series2];
   ans3 = Apply[Times, series3];
   Return[FullSimplify[SeriesCoefficient[ans1,r]
         SeriesCoefficient[ans2,s] SeriesCoefficient[ans3,t]]];)
```

Contribution 2

To compute the path integrals over the traces tr U, tr U^2..., it gives rise to the study of matrices that correspond to the kinetic energy factors for different number of contact interactions. For example, suppose there are two spin-up particles (named 1 and 2), and two spin-down particles (named 3 and 4). Also, suppose there is one contact interaction. Then there are 4 possibilities based on which pair of particles that are in contact: (1) particles 1 and 3, (2) particles 1 and 4, (3) particles 2 and 3, and (4) particles 2 and 4. Each scenario is called a Kop term; we name the 4 terms **k1a**, **k1b**, **k1c**, and **k1d** respectively. For two contact interactions, there are two Kop's: (1) **k2a**, the contact particles pairs are (1, 3), and (2, 4) resp. (2) **k2b**, the contact particles pairs are (1, 4), and (2, 3) resp. For 0 contact interaction, there is one Kop **k0**. For a given number **pairNum** of 2-body contact interactions, the following generate mechanically the different Kop's where **L1** and **L2** are two lists of spin-up and spin-down particles respectively.

```
def generateKopTerms(L1, L2, pairNum):
   if pairNum > min(len(L1),len(L2)):  return []
   if pairNum == 0:  return [[]]
   kTermList = []
   for i in range(len(L1)):
      for j in range(len(L2)):
         for L in generateKopTerms(L1[i+1:],L2[:j]+L2[j+1:],pairNum-1):
            kTermList += [[[L1[i],L2[j]]] + L]
   return kTermList

> generateKopTerms([1,2], [3,4], 0)
[[]]
> generateKopTerms([1,2], [3,4], 1)
[[[1, 3]], [[1, 4]], [[2, 3]], [[2, 4]]]
> generateKopTerms([1,2], [3,4], 2)
[[[1, 3], [2, 4]], [[1, 4], [2, 3]]]
```

For 3-body interaction, the codes become:

```
def generateKopTerms(L1, L2, L3, pairNum):
   if pairNum > min(len(L1),len(L2),len(L3)):  return []
   if pairNum == 0:  return [[]]
   kTermList = []
   for i in range(len(L1)):
      for j in range(len(L2)):
         for k in range(len(L3)):
            for L in generateKopTerms(L1[i+1:],L2[:j]+L2[j+1:],
                                      L3[:k]+L3[k+1:],pairNum-1):
               kTermList += [[[L1[i],L2[j],L3[k]]] + L]
   return kTermList

> generateKopTerms([0,1,2],[3,4],[5,6],2)
[[[0, 3, 5], [1, 4, 6]], [[0, 3, 5], [2, 4, 6]], [[0, 3, 6], [1, 4, 5]],
 [[0, 3, 6], [2, 4, 5]], [[0, 4, 5], [1, 3, 6]], [[0, 4, 5], [2, 3, 6]],
 [[0, 4, 6], [1, 3, 5]], [[0, 4, 6], [2, 3, 5]], [[1, 3, 5], [2, 4, 6]],
 [[1, 3, 6], [2, 4, 5]], [[1, 4, 5], [2, 3, 6]], [[1, 4, 6], [2, 3, 5]]]
```

For the general N-body interaction, we have:

```
def generateKopTerms(L, pairNum):
   if sum([pairNum > len(l) for l in L]) > 0:  return []
   if pairNum == 0:  return [[]]
   kTermList = []
   for i in range(len(L[0])):
      for (l1, L1) in generateAll(L[1:]):
         for L2 in generateKopTerms([L[0][i+1:]]+L1,pairNum-1):
            kTermList += [[[L[0][i]]+l1] + L2]
   return kTermList


def generateAll(L):
   if L==[]:  return [([],[])]
   ans = []
   for i in range(len(L[0])):
      for (x,y) in generateAll(L[1:]):
         ans.append( ([L[0][i]]+x, [L[0][:i]+L[0][i+1:]]+y) )
   return ans

> generateKopTerms([[0,1,2],[3,4],[5,6]],2)        # 3-body interaction
[[[0, 3, 5], [1, 4, 6]], [[0, 3, 5], [2, 4, 6]], [[0, 3, 6], [1, 4, 5]],
 [[0, 3, 6], [2, 4, 5]], [[0, 4, 5], [1, 3, 6]], [[0, 4, 5], [2, 3, 6]],
 [[0, 4, 6], [1, 3, 5]], [[0, 4, 6], [2, 3, 5]], [[1, 3, 5], [2, 4, 6]],
 [[1, 3, 6], [2, 4, 5]], [[1, 4, 5], [2, 3, 6]], [[1, 4, 6], [2, 3, 5]]]
```

Contribution 3

$\mathcal{U}=\mathcal{U}_1\mathcal{U}_2\ldots\mathcal{U}_{N_\tau}$ where $\mathcal{U}_i=\mathcal{T}\mathcal{V}_i\mathcal{T}$ using Trotter-Suzuki factorization, and $\mathcal{T} = e^{-\frac{\tau}{2}\hat{T}}$ is the kinetic energy factor for half-step imaginary time. Continue the previous example; suppose we have the Kop sequence **[k1d, k0, k2b, k1a]**, where the contact interactions at the 0th term **k1d** gives knot **0_0**, at the 2nd term **k2b** gives knots **2_0** and **2_1**, and at the 3rd term **k1a** gives knot **3_0**.

```
def out_T( N, Ntau, line, i, j, newline ):
   k = int(2*(j[0]-i[0]))
   if k == 1:              print("T[", end='')
   else:                   print(f"T{k}[", end='')
   if i[0] == -0.5:        print(chr(ord('a') + line-1) + ',', end='')
   else:                   print(f"{i[0]}_{i[1]},", end='')
   if j[0]==Ntau-0.5:      print(chr(ord('a') + N + line-1) + ']', end='')
   else:                   print(f"{j[0]}_{j[1]}]", end='')
   if newline:             print()
   else:                   print(' . ', end='')


N = 4
dict_ops={'k0':[], 'k1a':[[1,3]], 'k1b':[[1,4]], 'k1c':[[2,3]],
          'k1d':[[2,4]], 'k2a':[[1,3],[2,4]], 'k2b':[[1,4],[2,3]]}
terms = ['k1d', 'k0', 'k2b', 'k1a']
Ntau = len(terms)
states = [(-0.5,0)]*N
for tau, term in enumerate(terms):
   for idx, op in enumerate(dict_ops[term]):
      for line in op:
         out_T(N, Ntau, line, states[line-1], (tau,idx), False )
         states[line-1] = (tau,idx)
for line in range(1,N+1):
   out_T(N, Ntau, line, states[line-1], (Ntau-0.5,0), line==N)
```

The program outputs

```
T[b,0_0] . T[d,0_0] . T5[a,2_0] . T4[0_0,2_0] . T4[0_0,2_1] . T5[c,2_1]
 . T2[2_0,3_0] . T2[2_1,3_0] . T[3_0,e] . T3[2_1,f] . T[3_0,g] . T3[2_0,h]
```

which gives the trace $\sum_{i,j,k,l} [\mathcal{T}]_{bi}[\mathcal{T}]_{di}[\mathcal{T}^5]_{aj}[\mathcal{T}^4]_{ij}[\mathcal{T}^4]_{ik}[\mathcal{T}^5]_{ck}[\mathcal{T}^2]_{jl}[\mathcal{T}^2]_{kl}[\mathcal{T}]_{le}[\mathcal{T}^3]_{kf}[\mathcal{T}]_{lg}[\mathcal{T}^3]_{jh}$

Conclusion

Computing the virial coefficients analytically using automated algebra requires efficient codes. The research reported provides efficient and robust codes that support important functions in the front end of the computations. The main challenge will be addressed in a forthcoming paper.

Acknowledgement

I would like to thank Prof. Joaquin Drut and Yaqi Hou for their patience and excellent guidance.

References

[1] Y. Nishida and D. T. Son. *Universal four-component Fermi gas in one dimension*. ArXiv:0908.2159v3, 20 Oct 2010.

[2] J. R. McKenney, A. Jose and J. E. Drut. *Thermodynamics and static response of anomalous 1D fermions via quantum Monte Carlo in the worldline representation*. ArXiv:2003.01616, 3 Mar 2020.