ChiEFTint.jl:相互作用生成コード

Sota Yoshida

2021 年 10 月 20 日

カイラル有効場理論の相互作用を生成する Julia コード、ChiEFTint.jl について、概要や使い方をまとめる。
このコードは主として

- Entem-Machleidt 型の 2 体力 [1]
- 運動量空間での SRG(Similarity Renormalization Group) 変換
- Fermi gas 近似による有効 2 体化 3 体力 [2, 3, 4]
- valence 系の演算子を含む NN 相互作用 [5]
- IMSRG 計算を用いた 3 体力 LECs のベイズ最適化

などの計算が可能で運動量空間や殻模型計算用の相互作用ファイル (KSHELL/ShellModel.jl 用) を生成することができる[*1]。

1 Julia 環境の準備

Julia のバイナリをダウンロードするか、Unix(Mac)/Linux 環境なら wget などを使えばよい。
2021 年 10 月 16 日現在の最新版は v1.6.3 で、
`$wget https://julialang-s3.julialang.org/bin/linux/x64/1.6/julia-1.6.3-linux-x86_64.tar.gz`
を実行し展開した後、`julia-1.6.3/bin/julia` にエイリアスを貼るなどすれば良い。

1.1 REPL

上記の実行ファイルを実行すると、Julia の REPL(対話的な実行環境) が開く。
Python の対話モードのように使える。

1.2 パッケージ

Julia には Python で言う pip のようなパッケージ管理システムが用意されている。
REPL 内で] を押すと、pkg モードになり `$add packagename` で任意の公式パッケージを導入できる。
特に指定しなければ通常 home ディレクトリの直下に `.julia` というフォルダが作成され、自身が導入したパッケージなどが入る。
スパコンの計算ノードなどで自前の Julia を使う際、計算ノード上からみたパス `~/` にアクセス権限がない場合がある。その場合は、計算ノードから見える working ディレクトリなどに `.julia` を置いておいて `exportJULIA_DEPOT_PATH="/work/xx/yy/.julia"` を実行するなりジョブスクリプトに書いて置けばよい。
ちなみに `~/.julia/environments/v1.6/Project.toml`, `~/.julia/environments/v1.6/Manifest.toml` に、インストールしたパッケージの名前と UUID, 依存関係がまとめられている。
自作パッケージ XX を作る際には `Project.toml`, `Manifest.toml` を置くことで、XX を使う際には自動で開発

[*1] ココで言う「殻模型計算の相互作用ファイル」は自由空間の相互作用の意味で、基本的に、殻模型でそのまま使ってもしょうがない。模型空間用の相互作用がほしければ、MBPT(EKK) や VS-IMSRG など別途計算が必要。

者の環境が再現でき依存関係等をユーザーが気にしなくても良いようになっている。

また、任意のディレクトリに独立した Julia のパッケージ環境を構築することもできる[*2]。

2 ChiEFTint.jl の概要

2.1 準備

GitHub からクローンする[*3]

`$git clone https://github.com/SotaYoshida/ChiralEFTint.jl`

次に

`$cd ChiEFTint.jl`, `$julia src/package_install.jl`

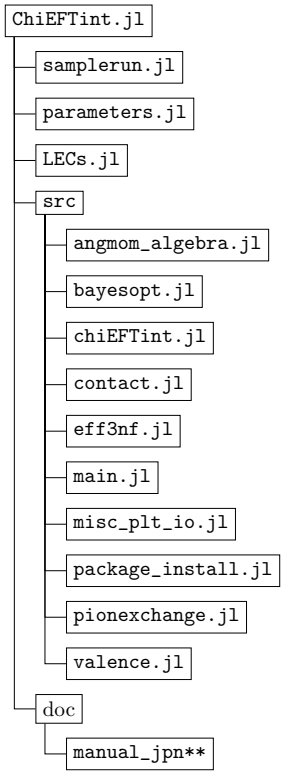
を実行し使用しているパッケージをインストールする。[*4]

`$julia -t 12 samplerun.jl` などと実行すると相互作用ファイルの生成ができる (はず)[*5]。

2.2 インプット/ソースファイルの概要

`ChiEFTint.jl/samplerun.jl` に引数無しの `make_chiEFTint()` という関数がある。`make_chiEFTint()` の実行時には、`ChiEFTint.jl/parameters.jl` で指定したパラメータ ($e_{\max}$ や $\hbar\omega$ など) が使用される。ユーザーは主に `ChiEFTint.jl/parameters.jl` のみを編集すれば良い。また coupling constants/Low-energy constants (LECs) は `ChiEFTint.jl/LEC.jl` にある[*6]。

それぞれのファイルの簡単な説明やディレクトリ構造は図 1 の通り:



`angmom_algebra.jl`	角運動量関連や HOB(Talmi 変換) の計算
`bayesopt.jl`	IMSRG+ ベイズ最適化のための部分
`chiEFTint.jl`	モジュールファイル (使用パッケージの指定, 関数の export)
`contact.jl`	contact 項の寄与の計算
`eff3nf.jl`	有効 2 体化 3 体力の計算
`main.jl`	main 部分
`misc_plt_io.jl`	お絵かきや I/O 関係
`package_install.jl`	使用パッケージのインストール用
`pionexchange.jl`	pion 交換項の寄与の計算
`valence.jl`	バレンス系演算子の寄与の計算

図 1 階層構造と src ファイルの説明

[*2] Julia in Physics2021 における佐藤建太氏の講演動画がわかりやすい

[*3] 2021 年 10 月 20 日現在はコードは public ではなく欲しい方にのみ提供、としている

[*4] 本来はパッケージ化して前述の Project/Manifest.toml で自動で依存関係を整えるのが望ましいが簡単のためこの方法を採用している

[*5] Julia ではコマンドラインで-t オプションをつけるだけで使用するスレッド数を指定することが出来る

[*6] const を使って global 変数として与えている。一方で、実際の寄与を計算する各種関数の中では LECs という辞書の値を参照して使っている。LECs を自分で最適化したりしたければ、辞書の対応するものの値を更新するように使えば良い。

3 ユーザーが指定すべきパラメータ

`parameters.jl` の中にあるパラメータのうち、通常の用途で変更すべきものは以下の通り

変数名	値の例	説明
`n_mesh`	40	運動量メッシュ点の数 [a]
`emax`	8	harmonic oscillator の major shell の数
`chi_order`	3	カイラル摂動論の次数 (正確には非ゼロの寄与をする次数, N3LO=3)
`calc_NN`	true/false	Entem-Machleidt 型 2 体力の計算
`calc_3N`	true/false	有効 2 体化 3 体力の計算
`kF`	1.35	有効 2 体化 3 体力の計算に使う Fermi momentum
`hw`	20.0	harmonic oscillator のパラメータ $\hbar\omega$
`Anum`	4	質量数 ($\hbar\omega$ を公式で決めずに手で入れるなら使用しない)
`srg`	true/false	運動量空間での SRG 変換を行うかどうか [b]
`srg_lambda`	2.0	SRG 変換のパラメータ $[fm]^{-1}$ [c]
`tbme_fmt`	`"snt"/"myg"`	相互作用フォーマット [d]
`target_nlj`	[ ]	興味のある軌道 ($[[n,l,j]]$) のリスト [e]
`v_chi_order`	0	(NLO=1) バレンス系の演算子の項を計算 [f]

[a] TBME 生成には 40 もあれば精度は十分だが、運動量空間で見栄えの良い図が書きたければ 100-200 にすると良い。ただし、その分 SRG 変換などは遅くなる。

[b] 今の実装では induce される多体力は考慮していない (いわゆる NN-only に相当)

[c] IMSRG などの文献では良く $s = 1/\lambda^4$ が用いられる

[d] snt 形式 (KSHELL/ShellModel.jl のインプット) か、myg(Takayuki Miyagi) 形式

[e] 該当するものだけをトリミングしてくれるはずだが... 限られた例でしか確認していないので注意が必要

[f] 未チェックなので、使用は非推奨, N3LO=3 は未実装

4 LECs

`ChiEFTint.jl/LECs.jl` には Low-energy constants がまとめられている。

2 体力については Entem Machleidt のレビュー [1] にある値を使用しているが、NNLO3 体力で初めて現れる c_D, c_E については適当な値を取っている。

とくに 2 体力は基本固定して使うことを想定しているが、前述のように LECs を自分で最適化したりしたければ、`ChiEFTint.jl/src/main.jl` を書き換えれば良い[*7]。より具体的には、`ChiEFTint.jl/src/main.jl` 内の `make_chiEFTint` 関数の後半部分に、IMSRG 計算を用いたベイズ最適化のコードが書いてあるので、それに倣って別の方法で最適化する関数を挿し込めば良い。

参考文献

[1] R. Machleidt and D. Entem, Phys. Rept. **503**, 1 (2011).

[2] M. Kohno, Phys. Rev. C **88**, 064005 (2013a).

[3] M. Kohno, Phys. Rev. C **96**, 059903 (2017b).

[4] S. Yoshida, M. Kohno, T. Abe, T. Otsuka, N. Tsunoda, and N. Shimizu, JPS Conf. Proc, **23**, 013014 (2018a).

[*7] induce される多体力を無視しているので、むしろ $\mathrm{NNLO_{sat}}$ のような哲学で LECs を決めるのもよいかもしれない

[5] L. Huth, V. Durant, J. Simonis, and A. Schwenk, Phys. Rev. C **98**, 044301 (2018b).