Hofstadterの蝶

志村昂輝

2026年1月14日

一級品として表れる諸々の技術や芸当は、その自由自在さで我々を驚かせますが、その背後には素人の知りえない膨大なルールと型があります。研究も例外ではなく、よりよい研究で発揮される個性はまずある一定の型やルールを身に着けた後に生まれています。研究における型を身に着ける最初の手段は、例えば学部生のうちには標準的な教科書を読んで知識を増やすことでしょうが、これは全くもって序の口にすぎず、一人前となるためには配属された研究室で師弟関係を結び、その研究室が売りとするスタイルを吸収していかなければなりません。

問題となるのが、この研究室の売りは一朝一夕に掴めるものではないということです。まず師弟関係の下で、弟子として師匠の謦咳に接しているかどうかが、素人とそうでない人とを分ける最初の分水嶺となります。将来自分の同業者になると見込んだ人間に対して、師匠はその分野の共通言語や常識を叩き込みますが、一人前とみなされるにはその量と厳しさをもってしても数年を要します。修士と博士合わせて5年かかるのは、この修行が並大抵のものではなく、素人から業界人へと決定的に変化させる重大なプロセスであることを示唆しているでしょう。

本稿は以上2つの困難を克服するために作成されています。つまり、加藤研内部で共有されている重要だがインアクセシブルな技術をまとめ、研究を始める際に典型的に出くわす困難とその対処法を示すこと、及び内容を極力端的に整理し、一学生から研究者になるための経路を効率よく整備することです。ただし、私の特殊なバックグラウンドに起因する限界について読者にお断りしなくてはなりません。まず加藤雄介研究室で扱われるテーマと技術は広く、私がここで示す内容はそのほんの一部にすぎないことです。もとより本研究室の強みは数値計算よりも微分方程式の求解や特殊関数の操作といった解析計算に存在しており、本稿の内容はむしろ傍流といえるかもしれません。主流といえるそちらの技術については、読者の皆様が教員との長い議論や共同研究の中で少しずつ身に着けたり、あるいは今後誰かが同様にドキュメント化してくれることを期待いたします。

第1章では、数値計算や解析の作業に適したプログラミング言語の紹介と簡単な文法の解説、及び環境構築の仕方について述べます。

強相関物質の研究スタイルにはかなり決まった型が存在していますが、出発点として死活的に重要なのがハミルトニアンの構築とその対角化です。第2章では、強束縛模型について軽く説明した後、ハミルトニアンを実装する方法、及びその対角化の方法について述べます。ハミルトニアンの対角化により得られた固有値がエネルギー分散ですが、実際に研究したり論文を書いたりするうえで必要なバンド図やフェルミ面の書き方についてもここで解説します。

第3章ではグリーン関数の実装方法について述べます。グリーン関数の虚部から得られる状態密度の描画方法もここで述べます。

第1章　プログラミング言語の選択と環境構築

1.1　数値計算向きの言語 (C++, Fortran)

数値計算という目的に絞ってもプログラミング言語の選択肢は多岐にわたりますが、物性物理で格子模型や多体問題を扱う場合、計算量は系の大きさや自由度の増加とともに急激に増大します。したがって、実行速度やメモリ配置を意識する必要があり、C++やFortranはそれにふさわしい選択肢の一つとして挙げられます。明示的に型が指定されるために、数値誤差のふるまいを把握しやすくなるのも重要です。

1.1.1　C++について

C++はC言語が基盤となっていますが、クラスやテンプレートといった機能を導入することで大規模なプログラムの構築や保守に向いています。本稿で主に用いる言語です。コンパイラもフリーで手に入り、特にLinuxでは標準装備されています。Windowsでもコンパイラがフリーで入手できるようです。

数値計算上一番大きなメリットはなんといっても、実行速度の速さです。C++はコンパイル型言語で、ループなどが機械語に近い形で最適化されるために、行列演算や反復計算で高い性能を発揮します。また配列の確保や解放を動的に行えば巨大配列を扱うこともできるので、波数空間上の物理量の情報を格納する場合に非常に効率がよいです。数値計算のライブラリも充実しています。

デメリットとして、メモリ管理を意識した低水準な記述が前提となっているために、数値計算にバグが混じりやすくなることが挙げられます。Fortranに比べると覚える概念が多く、特にポインタは大部分の学習者が躓く場所として悪名高いもので、言語に対する正確な理解と注意深い実装が求められます。それでも私がここでC++をお勧めするのは、学習コストが高い分ほかの言語を学習する際のハードルが下がるであろうことや、Fortranに比べるとできることが多いこと、最後に身も蓋もありませんが、筆者がFortranよりも長く触れているためにドキュメントを作りやすいことがあります。例えば個人的に文字列操作はFortranよりもC++のほうが便利だと考えています。

1.1.2　Fortranについて

数値計算の言語としてはFortranも依然重要な選択肢といえます。物性理論だけでなく、科学技術計算に特化した言語として長い歴史を持つために、研究室によっては保守性

の観点から Fortran によるコーディングを強いる場合もあります。これはオリジナリティ保護の観点から理にかなっていて、研究室の強みが Fortran に支えられており、いわば研究室独自のライブラリとして使いまわすことができるのです。大規模プロジェクトならなおさら、既存の技術的な資産を一人で書き換えることも不可能でしょう。しかしながら加藤雄介研究室は数値計算の研究室ではなく、そのような蓄積はないので絶対に Fortran を使わなければいけない環境ではありません。

Fortran にはポインタなどの概念が存在せず、比較的コーディングしやすいのも利点です。数値計算以外には基本的に向いていないのがデメリットで、Fortran ができるよりも C++ができる人のほうが、今後数値計算以外のことを仕事にする場合に融通が利くのではないかと考えています。

1.1.3 Python3 について

実のところ筆者が一番長く触れている言語です。数値計算を補助する言語として Python3 も非常に有用です。C++や Fortran に比べると、Python3 はライブラリが充実していて、可視化やデータ解析を迅速に行える利点もあります。[1] 型指定もないので、簡単な計算を行いたいだけなら C++や Fortran よりも圧倒的に便利です。特に配列 (リスト) の定義が楽で、線形代数関連のライブラリが充実しており簡単に計算できるのは大きな魅力です。コードの読解も比較的やさしく、精神的な負担が少ないです。

Python3 はスクリプト言語であり、実行速度やメモリ制御の面ではコンパイル型言語に劣るために、大規模数値計算の中核を担うのには不適切である場合もあります[2]。また型宣言がないのは実装や保守性の観点から苦しみの種となる場合もあります。しかしながら、C++や Fortran で実行した本計算の出力を Python で読み込み、プロットや解析を行うといった使い方は広く行われており、その自由度はほかの言語の追随を許しません。

1.1.4 近年注目されている言語 (Julia)

筆者は常用しておらず、軽く触れる程度しかできませんが、近年数値計算を目的としたプログラミング言語として注目されているものの一つに Julia があります。素早く実装することができ、かつ可読性が高い部分は Python に似ていますが、型安定なコードなら C や Fortran に劣らない速度で計算できるため、両者のいいとこどりをしています。現時点でのデメリットといえば、比較的新しい言語であるためにライブラリやパッケージの進化が早くバージョン管理が大変であろうことだと考えられますが、最近は研究会なども開かれており今後利用者は増えていくと見込まれます。

[1] 可視化には matplotlib, 数値計算やデータ解析には numpy や scipy といったライブラリが便利でしばしば使われます。

[2] 厳密には計算効率を上げるための工夫が様々に存在するようですが、その工夫に割くコストを考えると最初から C++や Fortran で実装した方がよいという意見が専門家の間では散見されます。

1.2 C++で使える高速数値計算ライブラリ

行列の積は添え字をループで回して和を取れば計算できますし、フーリエ変換もただの数値積分として実装することは一応できます。しかしそれらを自前で実装するのは時間がかかりますし、何より思わぬバグを招いてなかなか研究が進まないといった問題に直面しえます[2]。物性理論で頻出するこうした計算は、標準的なライブラリを呼び出して使うのが一般的です。代表的なものを以下に列挙しますが、実装例はのちの章で軽い文法とともに解説します。

1.2.1 BLAS/LAPACK(線形代数用ライブラリ)

BLAS はベクトルおよび行列に対する基本的な演算を提供する数値線形代数のライブラリで、ベクトルの内積をはじめ行列とベクトルの積、行列と行列の積も計算することが出来ます。これらの演算はそれぞれ Level 1, 2, 3 のように分類されています。LAPACK は BLAS を基礎として、行列の対角化や連立一次方程式の求解などを行うことができます。LU 分解や QR 分解などの行列の分解を行うこともできます。これらの外部ライブラリは時と試行回数の試練を潜り抜け、正確性を保証した業界のデファクトスタンダードとなっています。

1.2.2 FFTW3(フーリエ変換)

高速フーリエ変換 (FFT) は読んで字のごとく、フーリエ変換を高速に行うためのライブラリです。多次元のフーリエ変換も行うことが出来ます[1]。関数にはいろいろあり、実数関数と複素関数で使い分ける必要があります。また逆フーリエ変換とフーリエ変換の係数の違いは反映していないので、こちらで指定する必要があります。

1.2.3 OpenMP(並列計算)

ループ計算を行うとき、メモリ並列を行って複数の計算を同時に行うことで時間が短縮できることがあります。その手段として OpenMP が用いられます。並列化には MPI という手段もありますが、MPI はプロセスごとに情報の通信を行う必要があり、やや学習コストが高いものとなっています。スレッド並列なので既存のコードをほとんど変えずに並列化可能なのが利点とも言えます。

---

[2]勉強のために一から実装するのは効果的ではありますが、修士博士を途中まで経た身からすると、そうしたことにかまけている時間はそれほどなくお勧めできません

[1]昔はメッシュサイズが 2 の冪でないと計算できなかったようですが、アルゴリズムの進歩により任意のメッシュサイズで計算できるようになっています。

第2章　C++による数値計算の基礎

2.1　C++の文法速習

2.2　C++による対角化

本節では、C++からLAPACKを呼び出してエルミート行列を対角化するための手順をまとめます。複素行列を扱うか実対称行列を扱うかで用いるLAPACKのルーチンが異なる点にも注意すべきですが、呼び出し自体は数行で済みます。対角化には、エルミート行列を対角化するzheevか、dsyevを用います。LAPACKはもともとFortranで用いられていたライブラリであるため、column-majorでメモリの連続領域に格納されています。そこでC++では`std::vector<std::complex<double>>`を用いることで、要素H_{ij}に$H[i+N*j]$としてアクセスすることにします。

2.2.1　zheevの引数

zheevの引数はこちらが考えて指定するものはそれほど多くなく、慣れれば簡単に実装できるようになりますが、一応各引数の説明をします。

- `jobz`：char型。'V'か'N'の値をとる。'V'では固有値と固有ベクトルを計算する。'N'では固有値のみを計算する。
- `uplo`：char型。'U'か'L'の値を取る。エルミート行列は上三角部分と下三角部分さえ参照すれば全体の情報が分かるので、どちらの部分を参照するかを指定する。
- `n`：int型。対角化する行列のサイズを入力する。3×3行列なら3と入力する。
- `a(n*n)`：`std::vector<std::complex<double>>`型。入力として、対角化したい行列を与える。
- `lda`：int型。普通はnでよい。これは1次元配列として行列を考えたときに、次の列の情報をメモリのどこに渡すかを指定するものなので、nより大きな値を指定しても動く場合が多い。むしろnより小さな値を与えると意図しない結果をもたらすことになる。
- `w`：`std::vector<double>`型。行列の固有値を昇順に返す。エルミート行列の固有値はすべて実数であるので、double型としてよい。

- `lwork` : int 型で、最初に-1 を指定する。対角化しようとしている行列サイズで作業配列がどれくらい必要かを LAPACK に問い合わせるための引数。
- `work` : `std::complex<double>*`型。lwork で取得したサイズで work を確保する。
- `rwork` : `std::vector<double>`型。ふつうは `3*n-2` で指定する。
- `info` : int 型。対角化の一連の作業が正常終了したかどうかを表し、例外処理などに用いる。info が 0 なら正常終了しているが、それ以外の場合はどこかで失敗している。info が正の値であると、−info 番目の引数に不正があったことが示される。info が正の場合は固有値計算が収束しなかったことを示す。

zheev は入力で与えた行列を上書きするので、元のハミルトニアンを別に使いたい場合は、対角化前に行列をコピーして保持しておくべきです。

2.2.2 対角化の手順

ここでは

$$H = \begin{pmatrix} 1.0 & 0.2+0.1i & 0 \\ 0.2-0.1i & 2.0 & 0.3 \\ 0 & 0.3 & 3.0 \end{pmatrix} \tag{2.1}$$

を対角化するコードを書いてみましょう。この行列の固有値は

$$\lambda_1 \approx 0.95028847, \qquad \lambda_2 \approx 1.96487426, \qquad \lambda_3 \approx 3.08483726.$$

となっていて、解析的にはきれいに求まりません。手できれいに答えが分かるような場合はチェックが簡単ですが、実地の研究では圧倒的に手で解けない場合の方が多いので、このようなケースを扱います。

きちんと対角化できているかをチェックするには、エルミート行列の固有値及び固有ベクトルの性質に問題がないかを確かめる必要があります。以下のような性質があります。

- 固有値は実数である
- 固有ベクトルは正規直交系をなす

固有ベクトルが正規直交基底をなすということは、固有ベクトルを並べた行列 V が $V^\dagger V = \mathcal{I}$ を満たすということですから、これを計算することによって対角化が正しくできているかを確認できるといえるでしょう。

2.3 C++による FFTW3 を用いたフーリエ変換

次に FFTW を用いたフーリエ変換の解説に移ります。使用には`<fftw3.h>`をインクルードする必要があります。コンパイル時にもリンクを指定する必要があり、`-lfftw3` とします。

2.3.1 fftw_plan fftw_plan_dft_2d の引数

- `n0`：int 型。第 1 次元の格子点の数。
- `n1`：int 型。第 2 次元の格子点の数。
- `in`：fftw_complex*型。入力となる複素数配列を示す。実部と虚部を切り分ける必要があり、ある数 a に対して `in[ix * n1 + iy][0] = a.real(); in[ix * n1 + iy][1] = a.imag();` を指定しなければならない。
- `out`：fftw_complex*型。出力用の複素数配列。入力の配列が破壊されてしまうことを許容すれば `in == out` としてもよい。
- `sign`：int 型。指定可能な値は `FFTW_FORWARD` か `FFTW_BACKWARD` の 2 つである。それぞれ順変換と逆変換に対応する。逆変換後には正規化を行うべし。
- `flags`：unsigned 型。`FFTW_ESTIMATE` と `FFTW_MEASURE` の 2 つがある。

2.3.2 フーリエ変換の手順

ここでは、実空間表示の関数を波数空間の関数にフーリエ変換する関数を作ってみましょう。実空間表示の関数として、ここでは 2 次元格子上ローレンチアン

$$f(x,y) = \frac{\delta^2}{x^2+y^2+\delta^2} \tag{2.2}$$

を採用します。2 次元フーリエ変換

$$F(k_x,k_y) = \sum_{x,y} e^{-i(k_x x + k_y y)} f(x,y) \tag{2.3}$$

を FFTW で数値計算して波数空間上にプロットします。なお上のようなローレンチアンのフーリエ変換は厳密解が知られていて

$$F(k_x,k_y) = e^{-\delta|k|} \tag{2.4}$$

です。

2.4 C++による OpenMP を用いた並列計算

関連図書

[1] Hofstadter, Douglas R., Energy levels and wave functions of Bloch electrons in rational and irrational magnetic fields, Phys. Rev. B **14**, 2239(1976).

[2] C R Dean 1, L Wang, P Maher, C Forsythe, F Ghahari, Y Gao, J Katoch, M Ishigami, P Moon, M Koshino, T Taniguchi, K Watanabe, K L Shepard, J Hone, P Kim, Hofstadter's butterfly and the fractal quantum Hall effect in moiré superlattices, Nature 497(7451), (2013)