JAEA-Data/Code
2024-007
DOI:10.11484/jaea-data-code-2024-007

JAEA-Data/Code

汎用炉心解析システム MARBLE3 の開発

Development of the Versatile Reactor Analysis Code System, MARBLE3

横山 賢治　羽様 平　谷中 裕　大木 繁夫

Kenji YOKOYAMA, Taira HAZAMA, Hiroshi TANINAKA and Shigeo OHKI

大洗研究所
高速炉サイクル研究開発センター
高速炉解析評価技術開発部

Fast Reactor Life-Cycle Safety and Integrity Evaluation Technology Development Department
Fast Reactor Cycle System Research and Development Center
Oarai Research and Development Institute

October 2024

Japan Atomic Energy Agency

日本原子力研究開発機構

国立研究開発法人日本原子力研究開発機構　研究開発推進部　科学技術情報課
〒 319-1112　茨城県那珂郡東海村大字村松 4 番地 49
E-mail: ird-support@jaea.go.jp

JAEA-Data/Code 2024-007

汎用炉心解析システムMARBLE3の開発

日本原子力研究開発機構　大洗研究所
高速炉サイクル研究開発センター　高速炉解析評価技術開発部

横山　賢治、羽様　平[+]、谷中　裕、大木　繁夫

(2024 年 5 月 27 日受理)

汎用炉心解析システム MARBLE の第 3 版である MARBLE3 を開発した。MARBLE の開発ではオブジェクト指向スクリプト言語 Python を用いており、これまでの開発では Python バージョン 2（Python2）を用いていたが、Python のバージョンアップの後方非互換性の問題により、Python の最新版である Python バージョン 3（Python3) では、MARBLE を動作させることができなくなっていた。このため、MARBLE3 の開発では全面的に改修を行って、Python3 で動作するように整備した。また、MARBLE3 では、新しく開発された解析コードのカプセル化や新しく提案された計算手法等の組み込みを行うとともに、メンテナンス性や拡張性、柔軟性の観点からユーザインターフェースの拡張やソルバーの再実装等を行った。

MARBLE3 では、新規に開発された 3 次元六角/三角体系輸送計算コード MINISTRI Ver.7（MINISTRI7）と 3 次元六角/三角体系拡散計算コード D-MINISTRI を利用できるように整備した。これらのコードは、MARBLE のサブシステムである核特性解析システム SCHEME や高速炉燃焼解析システム OPRHEUS で利用できる。また、MARBLE に組み込まれている炉心解析システム CBG のユーザインターフェースを拡張して、CBG の 2 次元 RZ 体系の拡散計算ソルバーや輸送計算ソルバーを SCHEME 上で利用できるように整備した。

一方、計算手法についても改良を加えた。MARBLE3 では、チェビシェフ有理関数近似法に基づく燃焼計算手法の改良に関する論文やミニマックス多項式近似法に基づく燃焼計算手法に関する論文で提案された計算手法を利用できるように、燃焼計算ソルバーの機能拡張を行った。また、メンテナンス性の観点から、MARBLE2 で導入された一点炉動特性ソルバー POINTKINETICS を廃止して、MARBLE3 では KINETICS ソルバーとして新たに整備し直した。

大洗研究所：〒 311-1393 茨城県東茨城郡大洗町成田町 4002
[+] 敦賀総合研究開発センター 高速炉プラント技術開発部

JAEA-Data/Code 2024-007

Development of the Versatile Reactor Analysis Code System, MARBLE3

Kenji YOKOYAMA, Taira HAZAMA+, Hiroshi TANINAKA and Shigeo OHKI

Fast Reactor Life-Cycle Safety and Integrity Evaluation Technology Development Department
Fast Reactor Cycle System Research and Development Center
Oarai Research and Development Institute
Japan Atomic Energy Agency
Oarai-machi, Higashiibaraki-gun, Ibaraki-ken

(Received May 27, 2024)

The third version of the versatile reactor analysis code system, MARBLE3, has been developed. In the development of the former version of MARBLE, object-oriented scripting language Python (Python2) had been used and then the latest version of Python (Python3) was released. However, due to its backward incompatibility, MARBLE no longer worked with Python3. For this reason, MARBLE3 has been fully modified and maintained to work with Python3. In MARBLE3, newly developed analysis codes and newly proposed calculation methods were incorporated, and the user interface was extended and solvers were reimplemented for maintainability, extensibility, and flexibility.

In MARBLE3, the three-dimensional hexagonal/triangular transport code MINISTRI Ver.7 (MINISTRI7) and the three-dimensional hexagonal/triangular diffusion code D-MINISTRI are available as the new analysis codes. These codes can be used in the neutronics analysis system SCHEME and the fast reactor burnup analysis system OPRHEUS, which are the subsystems of MARBLE. In addition, the user interface of CBG, a core analysis system embedded in MARBLE, was extended so that the diffusion and transport calculation solvers for the 2-dimensional RZ system of CBG can be used on SCHEME.

On the other hand, MARBLE3 has extended the functionality of the burnup calculation solver so that it can use the numerical methods proposed in the papers on the improvement of the Chebyshev rational function approximation method and the minimax polynomial approximation method. From the viewpoint of maintainability, the point reactor kinetics solver POINTKINETICS, which was introduced in MARBLE2, has been newly reworked as the KINETICS solver in MARBLE3.

Keywords: Fast Reactor, Neutronics, Nuclear Design, Object-oriented Technique, MARBLE, SCHEME, ORPHEUS

+ Fast Reactor Plant Technology Development Department, Tsuruga Comprehensive Research and Development Center

目 次

Contents

List of Tables

List of Figures

This is a blank page.

1. はじめに

日本原子力研究開発機構（原子力機構）では、柔軟性、再利用性、拡張性、保守性といった性能を備えた「工学系モデリング言語としての次世代解析システム」に関する検討を 2002 年から行ってきた [1–6]。これらの検討結果をふまえて、2011 年に高速炉核特性解析のための新しい解析システムとして、MARBLE (Multi-purpose Advanced Reactor Physics Analysis System Based on Language of Engineering）の第 1 版 (MARBLE1) を開発した [7]。その後も開発を進め、2015 年に第 2 版（MARBLE2）を整備した [8]。本報告書では、MARBLE2 以降に改良・整備した第 3 版（MARBLE3）について説明する。

MARBLE1 では、高速炉の核特性解析における主要な解析機能が使えることを目標に、従来の高速炉核特性解析システム JOINT-FR[9,10] の解析機能を統一的なユーザインターフェースで使えるように MARBLE 上のサブシステムである核特性解析システム SCHEME として整備した。また、JOINT-FR では対応することができなかった高速炉実機の燃料交換を伴うような詳細な燃焼計算に対応できるシステムとして、MARBLE 上で高速炉実機燃焼解析システム ORPHEUS を新たに開発して整備した。MARBLE2 では、MARBLE1 では整備できなかった感度解析コード SAGEP[11] や 2 次元輸送／摂動計算コード TWOTRAN[12]、SNPERT[13] のカプセル化による解析コードの実行制御や入出力ファイルの読み取りや書き出しの機能の整備を行い、従来の高速炉核特性解析システムのほぼ全ての機能を SCHEME のインターフェースを通して利用できるように整備した。また、MARBLE2 では新たに衝突確率計算、拡散計算、感度計算等の解析機能に関する独自のソルバーの開発 [14] や、MARBLE1 で組み込まれた燃焼計算の解析機能に関するソルバーの改良を行った。その他、従来の高速炉核特性解析システムには含まれていなかった一点炉動特性解析に関するソルバー Pointkinetics[15] が開発され、MARBLE2 に組み込まれた。

MARBLE の開発では、オブジェクト指向スクリプト言語 Python を利用しており、MARBLE1、MARBLE2 の開発ではいずれも Python バージョン 2（Python2）を利用していた。Python は、基本的にはバージョンが上がっても主要な文法等は変わらず、古いバージョンで書かれたプログラムは新しいバージョンでも動作する後方互換性が保たれていた。しかしながら、Python バージョン 3（Python3）では、後方互換性が保たれない機能追加や変更が行われた。このため、Python2 で開発された MARBLE1、MARBLE2 は、Python3 では動作させることができなくなっていた。MARBLE3 では、この問題を解決するため、Python3 で動作するように全面的に改修を行った。

また、MARBLE2 以降に開発・改良された 3 次元六角（Hex-Z）/三角（Tri-Z）体系の輸送計算コード MINISTRI Ver.7（MINISTRI7）[16] や 3 次元 Hex-Z/Tri-Z 体系の拡散計算コード D-MINISTRI[16] を MARBLE のサブシステムである SCHEME や ORPHEUS から利用できるように整備した。MARBLE2 には、3 次元 Tri-Z 体系輸送計算コード MINISTRI Ver.1.1（MINISTRI1）[17] が導入されていたが、MINISTRI7 は、MINISTRI1 に比べて大幅な計算時間の短縮が図られている [16]。

一方で、MARBLE2 では、プログラミング言語 C++で開発された炉心計算システム CBG[18] が組

み込まれたが、MARBLE のサブシステム SCHEME から利用するインターフェースとしては、拡散計算ソルバー PLOS（3 次元 XYZ 体系オプションのみに対応）や 3 次元 XYZ 体系輸送計算ソルバー SNT 等の一部の機能にしか対応していなかった。MARBLE3 では、CBG の機能の組み込みを進め、拡散計算ソルバー PLOS の 2 次元円筒（RZ）体系オプションや 2 次元 RZ 体系輸送計算ソルバー SNRZ の組み込みを行って、MARBLE のサブシステム SCHEME 上で利用できるように整備した。更に、これらのソルバーを拡散摂動計算コード PERKY[19] と連携して使えるように PERKY コードの SHCEME 関数の改良を行った。これにより、MARBLE のサブシステム SCHEME 上で、高速炉の核特性解析でよく用いられる 2 次元 RZ 体系、3 次元 XYZ 体系、Hex-Z 体系、Tri-Z 体系に対する解析コードやソルバーの選択肢が増え、主要な解析を国産のコードやソルバーだけでも実行できるようになった。

燃焼計算手法に関連して、チェビシェフ有理関数近似法（CRAM: Chebyshev Rational Approximation Method）の改良版やミニマックス多項式近似（MMPA: Mini-Max Polynomial Approximation）法を適用した燃焼計算の解法が提案されたことを受けて、MARBLE3 の燃焼計算ソルバーにこれらの解法を組み込んだ。また、MARBLE2 では、C++の数値計算ライブラリを使って開発された一点炉動特性ソルバー Pointkinetics が導入されたが、利用している数値計算ライブラリのメンテナンス性やインストールの煩雑さの回避の観点から、Python と Python の数値計算ライブラリを使って書き直し、ほぼ同等の機能をもつ一点炉動特性ソルバー KINETICS として整備し直した。

第 2 章では、MARBLE3 における改良点について説明する。第 3 章では MARBLE3 で改良・追加された燃焼計算ソルバーや実装し直した一点炉動特性ソルバーについて説明する。

2. 改良点の内容

本章では、MARBLE2 から MARBLE3 における改良点の内容について説明する。

2.1 オブジェクト指向スクリプト言語 Python バージョン 3 への対応

前述のように、MARBLE はオブジェクト指向スクリプト言語 Python を使って開発されている。Python は現在では機械学習や深層学習の分野でも広く用いられており、比較的仕様の安定したプログラミング言語であるが、Python2 から Python3 へのバージョンアップの際に Python の基本設計を改良することを目的として、Python2 との後方互換性を持たない機能追加や仕様変更が行われた。このため、Python2 で開発されたプログラムは Python3 ではそのまま動作しなくなった。この影響により、MARBLE2 も Python3 では動作しなくなった。

このため、MARBLE3 の開発では大幅にソースコードを変更して Python3 で動作するように整備した。なお、Python2 のサポートは既に終了しており、今後、MARBLE3 を Python2 で動作させる必要はないと考えられるので、MARBLE3 を Python2 で動作させることは考慮していない。このため、MARBLE3 は Python2 では動作せず、MARBLE3 を利用する際には Python3 が必須となる。

この Python3 への対応で実施した書き換え作業は、基本的には、MARBLE3 の内部のソースコードの書き方の問題であり、ユーザには直接影響しないが、Python2 に比べて Python3 ではデータの型等が厳密に扱われるようになっており、この書き換え作業によって、MARBLE3 は MARBLE2 に比べてより堅牢な書き方に修正されていると考えられる。前述のように、基本的には、ユーザには直接影響しない内容であるが、MARBLE1、MARBLE2 で使っていたスクリプトを MARBLE3 で動作させた場合には、一部影響のある修正も含まれるため、Python3 への対応の書き換え作業の概要を以下にまとめる。

- Python2 では文字列とバイナリデータに区別がなくいずれも str オブジェクトで扱われ、ユニコード文字列は別途 unicode オブジェクトで扱われていた。これに対して、Python3 では文字列とバイナリデータが区別されるようになり、それぞれ、str オブジェクト、bytes オブジェクトで扱われるようになった。ユニコード文字列は str オブジェクトに統合された。MARBLE では、文字列、バイナリデータの両方のデータを扱うため、両者を区別するように全面的に書き換えた。また、ユニコード文字列も一部利用していたため、これらの統合も行った。
- Python3 ではハッシュ値が定義されていないオブジェクトを辞書オブジェクト（`dict`）のキーとして使えない、集合オブジェクト（`set`）の要素の比較等の仕様変更が行われたため、MARBLE の実装におけるハッシュ値の定義（`__hash__()` メソッドの定義）の厳格化を行った。

- Python2 では辞書オブジェクト（dict）に格納されるデータの順序は不定であったが、Python3 では順序が保存されるようになったため、この仕様変更を受け入れてテストを修正した。このため、一部の MARBLE3 のオブジェクトではデータの格納順序に関する挙動が変更されている。このため、計算結果を格納順に表示させるようなスクリプトについては MARBLE3 で挙動が変化する可能性がある。
- Python3 で廃止された文や関数（print 文、file()、cmp()、time.clock() 等) を Python2 と同等の動作をするように修正した。
- Python3 で挙動や仕様が変更された関数や演算子、比較（max()、range()、/、//、None オブジェクトとの比較等）を Python2 と同等の動作をするように修正した。
- Python3 に対応しなくなった外部ライブラリ（PyTC[20]、Blitz++[21,22]）を別のライブラリで同等の動作をするように書き換えた。PyTC は、高速炉燃焼解析サブシステム ORPHEUS の内部で利用されていたキーバリュー型データベース（Tokyo Cabinet）を Python から利用するためのライブラリである。このデータベースは ORPHEUS 内部で利用されているデータベースであり、基本的に ORPHEUS のユーザインターフェースを介して利用するため、この変更はユーザには直接影響しない。また、MARBLE3 では、PyTC の代わりに Python の標準ライブラリの dbm モジュールを使用するように変更されているので MARBLE3 では PyTC（Tokyo Cabinet）をインストールする必要がなくなった。一方、Blitz++は一点炉動特性ソルバー POINTKINETICS で使われていた C++の数値計算ライブラリであるが、MARBLE3 では Python の数値計算ライブラリ（NumPy[23]、SciPy[24]）を使って、一点炉動特性ソルバー KINETICS を再実装し、POINTKINETICS ソルバーは廃止したため、Blitz++ をインストールする必要がなくなった。

2.2 解析コード・ソルバーの導入と改良

2.2.1 最新版の 3 次元 Hex-Z/Tri-Z 体系輸送計算コード MINISTRI の導入

前章で少し述べたように、MARBLE2 には、3 次元 Tri-Z 体系輸送計算コード MINISTRI Ver.1.1（MINISTRI1）[17] が導入されていたが、2019 年に MINISTRI の改良版である MINISTRI Ver.7.0（MINISTRI7）が開発された [16]。MARBLE3 では、MINISTRI7 を MARBLE のサブシステムである核特性解析システム SCHEME、高速炉実機燃焼解析システム ORPHEUS で利用できるように整備した。

MINISTRI7 の開発では、収束性の問題を詳細に分析することで大型炉心体系への適用における収束性が向上しており、更に、初期拡散計算機能の導入と並列処理化が行われたことにより、従来のバージョンに比べて大幅な計算時間の短縮が図られている [16]。この MINISTRI7 の並列計算機能は、MARBLE3 の SCHEME、ORPHEUS からも利用できるように整備されている。なお、MINISTRI7 は、3 次元 Hex-Z 体系輸送計算コード MINIHEX の機能も統合され、3 次元 Tri-Z 体系と 3 次元 Hex-Z 体系の両方に対応している。MARBLE3 は、MINISTRI7 の 3 次元 Tri-Z 体系輸

送計算、3 次元 Hex-Z 体系輸送計算の両方に対応している。このため、3 次元 Hex-Z 体系輸送計算を行うために解析コードを MINISTRI から MINIHEX に切り替える必要はなくなっており、どちらのメッシュに対しても MINISTRI7 を利用することができる。

MARBLE2 でも 3 次元 Hex-Z 体系輸送計算コード NSHEX[17] を利用することができたが、収束性の問題等から、MARBLE2 では、計算メッシュを 3 次元 Hex-Z 体系から 3 次元 XYZ 体系に変換して、3 次元 XYZ 体系輸送計算コード TRITAC[25,26] で計算することが多かった。しかしながら、今後は MINISTRI7 を利用して、直接、3 次元 Hex-Z/Tri-Z 体系で輸送計算を実行することが現実的になると考えられる。高速炉の炉心解析では、3 次元 Hex-Z 体系、3 次元 Tri-Z 体系でモデル化することが多いので、この後、MINISTRI7 は、高速炉核特性解析の中核ソルバーとして利用されることが期待される。

2.2.2　3 次元 Hex-Z/Tri-Z 体系拡散計算コード D-MINISTRI の導入

MINISTRI7 では初期拡散計算機能の導入が行われたが、MINISTRI7 の開発の一環として、この拡散計算機能を単独で利用できるように D-MINISTRI というコードが整備された [16]。この D-MINISTRI についても、MINISTRI7 と同様に、MARBLE3 の SCHEME、ORPHEUS で利用できるように整備した。MARBLE2 で導入された拡散計算ソルバー DIFFUSION は 3 次元 Hex-Z 体系、Tri-Z 体系にも対応しているが、DIFFUSION ソルバーは Python で実装されており、計算体系が大きくなることの多い 3 次元 Hex-Z 体系、Tri-Z 体系では計算時間がかかるため、本格的な解析に適用するのは現実的ではなかった。このため、MARBLE2 では、3 次元 Hex-Z 体系、Tri-Z 体系の拡散計算を行う際には、CITATION-FBR コード [9,27] や DIF3D コード [28] を利用する必要があったが、MARBLE3 ではこれらに加えて D-MINISTRI を利用できるようになった。

2.2.3　CBG システムの拡散計算ソルバー PLOS の 2 次元 RZ 体系への対応

MARBLE2 では、CBG システムの拡散計算コード PLOS の組み込みが行われ、MARBLE のサブシステム SCHEME 上で利用できるようになっていたが、3 次元 XYZ 体系にしか対応していなかった。PLOS 自体は 2 次元 RZ 体系にも対応しているため、MARBLE3 では、SCHEME 上で PLOS を使った 2 次元 RZ 体系拡散計算を実行できるようにユーザインターフェースを拡張した。MARBLE3 では、他の解析コードの SCHME 関数と同様に、PLOS の SCHEME 関数（`plos()`）に RzMesh オブジェクトを入力することで PLOS を使った 2 次元 RZ 体系拡散計算を実行することができる。

2 次元 RZ 体系であれば計算体系はそれほど大きくならないので、Python で実装された DIFFUSION ソルバーを利用できるケースもあるが、PLOS は C++で実装されており、DIFFUSION ソルバーに比べると大幅に計算時間を短縮することができる。

2.2.4 CBG システムの 2 次元 RZ 体系輸送計算ソルバー SNRZ の導入

CBG システムには、2 次元 RZ 体系輸送計算ソルバー SNRZ、3 次元 XYZ 体系輸送計算ソルバー SNT が実装されているが、MARBLE2 では SNT の SCHEME 関数しか整備されていなかった。このため、MARBLE2 では、2 次元 RZ 体系の輸送計算を行う場合には、TWOTRAN コードや PARTISN コードを利用する必要があったが、MARBLE3 では、SNRZ の SCHEME 関数（`snrz()`）を追加し、SCHEME 上で SNRZ を利用できるように整備した。これにより、MARBLE3 の SCHEME では、CBG システムの SNRZ ソルバーを使った 2 次元 RZ 体系輸送計算が可能になった。なお、SNRZ の SCHEME 関数においても、PLOS や他の解析コードの SCHEME 関数と同様に、RzMesh オブジェクトを入力することで、SNRZ を使った 2 次元 RZ 体系輸送計算を実行することができる。

2.2.5 拡散摂動計算コード PERKY の SCHEME 関数の改良

MARBLE2 では、拡散摂動計算コード PERKY の SCHEME 関数は、拡散計算計算コード（CITATION-FBR、DIF3D）との組合せが固定されていた（以下、結合版 PERKY 関数）が、MARBLE3 では、拡散計算コードとして、PLOS、D-MINISTRI が導入されたので、拡散計算コードとの組合せを固定せずに利用可能な PERKY の SCHEME 関数（以下、分離版 PERKY 関数）を整備した。

具体的には、MARBLE3 では、分離版 PERKY 関数として以下の関数が導入された。

- `perky_for_beta_effective()`
- `perky_for_propmt_neutron_lifetime()`
- `perky_for_worth_mapping()`
- `perky_for_reactivity()`

分離版 PERKY 関数では、拡散計算コードのコード名を明示しない代わりに、拡散計算で得られた中性子束と随伴中性子束のオブジェクトを入力する。これにより、PLOS や D-MINISTRI と PERKY を組み合わせた計算が可能になっている。ただし、MARBLE がベースとしている従来の高速炉核特性解析システムでは、長年に亘って CITATION-FBR と PERKY の組合せで利用経験が重ねられてきた。この利用経験に比べると、新しい組合せでの利用経験は非常に少ないので、今後、利用経験を重ねて検証を行っていく必要がある。

2.2.6 拡散摂動計算コード DIFPERT の導入

拡散摂動計算コード DIFPERT は、原子力機構の前身の核燃料サイクル研究開発機構で 2000 年頃に開発されたコードである。DIFPERT は、PERKY と同等の計算機能をを持つ解析コードであり、PERKY に比べると入力データを簡素化することができる。具体的には以下の計算機能を有する。

- 実効遅発中性子割合

- 即発中性子寿命
- 反応度価値マップ（一次摂動）
- 反応度変化（一次摂動、厳密摂動）

MARBLE3 では、摂動計算に関する検証を主な目的として、DIFPERT コードを SCHEME 上で利用できるように整備した。MARBLE 上では SCHEME 関数として利用できるように整備されているため、DIFPERT の入力データをユーザが直接作成する必要はなくなっているが、1970 年代に開発された PERKY に比べると新しいコードであるので、今後は PERKY に代わる拡散摂動計算コードとして利用できると期待される。

ただし、MARBLE3 では、結合版 PEKRKY と同様のインターフェースを持つ、CITATION-FBR、DIF3D コードと連携して利用するための SCHEME 関数（結合版 DIFPERT 関数）の整備のみであり、分離版 PERKY 関数のような拡散計算コードとの組合せを固定せずに利用可能な DIFPERT の SCHEME 関数は整備されていない。

2.2.7 燃焼計算ソルバー BURNUP への新しい行列指数計算手法の導入

MARBLE の燃焼計算ソルバー BURNUP は、基本的には行列指数法に基づく燃焼計算ソルバーであり、MARBLE2 では行列指数法に基づくソルバーとして以下の解法に対応していた。

- テイラー展開法（ORIGEN2 コード [29] を参考にした実装）
- パーデ近似法（SciPy[24] のソルバーを利用した実装）
- 固有値分解法（SciPy[24] のソルバーを利用した実装）
- クリロフ部分空間法（2007 年のクリロフ部分空間法の論文 [30] に基づく実装）
- チェビシェフ有理関数近似法 CRAM（2010 年の CRAM の論文 [31] に基づく実装）

MARBLE3 ではこれらの解法に加えて以下の解法を追加した。

- チェビシェフ有理関数近似法 CRAM-PFD（2011 年の CRAM の論文 [32] に基づく実装）
- チェビシェフ有理関数近似法 CRAM-IPF（2016 年の CRAM の論文 [33] に基づく実装）
- ミニマックス多項式近似法 MMPA（2015 年の MMPA の論文 [34] に基づく実装）

MARBLE3 で追加された解法の詳細については次章で説明する。

2.2.8 一点炉動特性ソルバーの再実装

MARBLE2 では、一点炉動特性ソルバー POINTKINETICS[15] が導入されたが、このソルバーでは、Blitz++[21,22] という C++の数値計算ライブラリが利用されている。Blitz++は C++のテンプレートプログラミングの技術を応用した多次元配列に対する高性能の数値計算ライブラリであり、MARBLE2 の開発当時は活発に開発が進められていたが、現在は開発が停止している状態である。GitHub の Web ページ [22] によると、Blitz++は、1999 年のプログラミング言語 C++の ISO 標準

（C99）で開発されており、2011 年の C++の ISO 標準（C++11）の恩恵を得ることができなくなっているとのことであり、新たなプロジェクトで利用することは推奨されていない。このような状況から、Blitz++の開発が再度活発化することは期待できないと判断し、MARBLE のメンテナンス性の観点から Blitz++ライブラリの利用の廃止を検討した。MARBLE2 では Blitz++を利用しているのは一点炉動特性ソルバー POINTKINETICS だけあり、一点炉動特性ソルバーは C++を使わなくても、MARBLE で汎用的に利用している Python の数値計算ライブラリ NumPy[23]、SciPy[24] の機能を使って効率的なソルバーを開発できることが分かったため、NumPy、SciPy を使って一点炉動特性ソルバー KINETICS を新たに実装することにした。なお、MARBLE3 では、POINTKINETICS ソルバーは廃止されたので、MARBLE3 を利用する際に、Blitz++をインストールする必要はなくなった。

MARBLE3 で再実装された一点炉動特性ソルバー KINETICS（Kinetics モジュール）の詳細については次章で説明する。

2.2.9 燃焼感度解析システム PSAGEP の統合と改良

MARBLE3 では、メンテナンス性の観点から、燃焼感度解析コードシステム PSAGEP [35–37] の MARBLE への統合を行った。また、MARBLE への統合後に、PSAGEP の核分裂スペクトルの感度係数の核種毎の分離機能を拡張したので、この点についても説明する。

2.2.9.1 燃焼感度解析システム PSAGEP の統合

MARBLE2 では、核データの単位変化に伴う核特性の変化率である感度係数を計算するための感度解析コード SAGEP[11] を利用できるように整備されたが、SAGEP が計算できるのは燃焼を伴わない核特性に対する感度係数だけであった。一方、燃焼感度係数を計算するための解析コードシステムとしては、燃焼感度解析コードシステム PSAGEP が整備されていた。PSAGEP は、FORTRAN 言語で開発された燃焼感度解析コードシステム SAGEP-BURN[38–40] を Python で制御するように整備し直したシステムである。

FORTRAN で開発された解析コードを Python で制御するという考え方は、MARBLE と同じであるが、PSAGEP は MARBLE よりも先に開発されたシステムであり、PSAGEP で開発されたフレームワークを参考に、MARBLE の開発が進められた。このため、MARBLE のフレームワークと PSAGEP のフレームワークは似ている部分もあるが、MARBLE のフレームワークはその後、長年に亘って改良が続けられため、両者のフレームワークには互換性がなくなっていた。これらのコードシステムの維持管理や機能拡張を効率的に行っていくためには、両者のフレームワークを統合するのが望ましい。このため、PSAGEP のフレームワークを MARBLE に統合し、PSAGEP を MARBLE のサブシステム（marble.psagep）として整備し直した[1]。これにより、これまでの単独

[1]厳密には、MARBLE3 では、PSAGEP のフレームワークが MARBLE のフレームワークと完全に統合されたわけではなく、同様の機能の重複が残っている部分もあるが、開発者は MARBLE と同じ枠組みでテストやバージョン管理を行えるようになっている。また、MARBLE と PSAGEP の内部の問題であるので、ユーザには特に影響しない。

システムの PSAGEP の代わりに、MARBLE のサブシステムとして再整備された PSAGEP を利用できるようになった。

PSAGEP の利用方法については、後述する核分裂スペクトルの感度係数に関する改良以外には変更点はなく、PSAGEP の実装やテスト、サンプル問題の格納されている場所（ディレクトリ）の違いに留意すれば、MARBLE のサブシステムとして再整備された PSAGEP は、オリジナルの PSAGEP と同じように利用することができる。表 2.2.1 に、オリジナルの PSAGEP と MARBLE3 に再整備された PSAGEP の格納場所の違いを示す。

表 2.2.1　MARBLE3 にサブシステムとして再整備された PSAGEP の格納場所

	オリジナルの PSAGEP	MARBLE に再整備された PSAGEP
実装	PSAGEP/sagep	MARBLE/marble/psagep
テスト	PSAGEP/test	MARBLE/tests/psagep
テスト用データ兼サンプル問題	PSAGEP/data	MARBLE/data/psagep

2.2.9.2　燃焼感度解析システム PSAGEP の改良

オリジナルの PSAGEP では、核分裂スペクトルの感度係数については、従来の SAGEP コード同様に、Pu-239 と U-235 の主要な二つの核分裂性核種に分離することしかできなかったが、MARBLE に再整備された PSAGEP では、SAPEP コードと同様の機能拡張が行われ、核分裂性核種の核分裂率を重みとして、核分裂スペクトルの感度係数を分離する機能が導入されている。

オリジナルの PSAGEP の入力データでは、核分裂スペクトルの感度係数を Pu-239 と U-235 に分離するために図 2.2.1 のようなオプションを指定する必要があったが、このオプションの指定を省略すると、核分裂スペクトルの感度係数の分離が自動的に行われる。なお、MARBLE に再整備された PSAGEP においても、核分裂スペクトルの感度係数を Pu-239 と U-235 に分離するためのオプションは有効であり、図 2.2.1 のようにオプションを指定すると、オリジナルの PSAGEP と同様に、核分裂スペクトルの感度係数を Pu-239 と U-235 に分離する。

```
sddb = StandardSensitivityDividerDB()
sddb.set_weights(title=core_name, pu239=8.95832E-04, u235=9.74561E-06)
info_man.setSensitivityDividerDB(sddb)
```

図 2.2.1　核分裂スペクトルの感度係数を Pu-239 と U-235 に分離する PSAGEP の入力オプション

2.3 MARBLE3で利用できる解析コードとソルバー

2.3.1 従来システムの解析コードの利用

MARBLE3では、MARBLE2と同様に従来の高速炉核特性解析コードシステムに含まれる以下の解析コードのカプセル化が行われており、MARBLEのインターフェースを使って利用することができる。

- 超微細群格子計算コード SLAROM-UF[41)]
- 拡散計算コード CITATION-FBR[9,27)]
- 拡散摂動計算コード PERKY[19)]
- 3次元XYZ体系輸送計算コード TRITAC[25,26)]
- 3次元XYZ体系輸送摂動計算コード SNPERT-3D[42)]
- 3次元六角体系輸送計算コード NSHEX[17)]
- 2次元輸送計算コード TWOTRAN[10,12)]
- 2次元摂動計算コード SNPERT[13)]
- 感度解析コード SAGEP[11)]

なお、MARBLEのベースとなっている従来システムは、インターフェースプログラムと呼ばれるJOINTコードを基盤とした高速炉の核特性解析用コードシステム[9)]がベースになっている。

2.3.2 従来システムに含まれていなかった解析コード・ソルバーの利用

これまでの説明をまとめると、MARBLE3では新たに以下の解析コードやソルバーを利用できるようになった。

- 3次元六角/三角メッシュ輸送計算コード MINISTRI7[16)]
- 3次元六角/三角メッシュ拡散計算コード D-MINISTRI[16)]
- CBGシステムの拡散計算コード PLOS[43)]
  （MARBLE2は3次元XYZ体系のみに対応、MARBLE3は2次元RZ体系にも対応）
- CBGシステムの2次元RZ体系輸送計算コード SNRZ[43)]
- CBGシステムの3次元XYZ体系輸送計算コード SNT[43)]
- 拡散摂動計算コード DIFPERT

その他、MARBLE3にはMARBLE2と同様にPythonで実装された以下のソルバーも含まれる。

- 衝突確率計算ソルバー PIJ
- 拡散計算ソルバー DIFFUSION
- 摂動計算ソルバー PERTURBATION
- 感度計算ソルバー SENSITIVITY

- 燃焼計算ソルバー BURNUP
- 設計精度評価ソルバー UNCERTAINTY
- 一点炉動特性解析ソルバー KINETICS
  （MARBLE2 の POINTKINETICS を MARBLE3 では KINETICS として再整備）

また、MARBLE2 と同様に、MARBLE3 においても以下の解析コードのライセンスを別途保有していれば、これらのコードを MARBLE のインターフェースを介して実行することができる。

- 拡散計算コード DIF3D[28)]
- 輸送計算コード PARTISN[44)]

2.3.3 MARBLE3 のサブシステム SCHEME で利用可能な輸送・拡散計算コード

表 2.3.1 に MARBLE3 の SCHEME で利用可能な解析コード・ソルバーをまとめる。この表から、炉心解析でよく用いる体系（2 次元 RZ 体系、3 次元 XYZ 体系、3 次元 Hex-Z 体系、3 次元 Tri-Z 体系）については、PLOS、D-MINISTRI、SNRZ、SNT、TRITAC、MINISTRI7 を用いることで、国産の解析コード・ソルバーで対応できるようになったことが分かる。

表 2.3.1 SCHEME で利用可能な輸送・拡散計算コード

体系	手法	コード・ソルバー
1 次元球体系	拡散	DIFFUSION / DIF3D
	輸送	PARTISN
1 次元スラブ体系	衝突確率	PIJ
	拡散	DIFFUSION
	輸送	PARTISN
1 次元円柱体系	衝突確率	PIJ
	拡散	DIFFUSION
2 次元 XY 体系	拡散	DIFFUSION
2 次元 Hex 体系	拡散	DIFFUSION
2 次元 Tri 体系	拡散	DIFFUSION
2 次元 Rθ 体系	拡散	DIFFUSION
2 次元 RZ 体系	拡散	DIFFUSION / PLOS / CITATION-FBR / DIF3D
	輸送	SNRZ / TWOTRAN / PARTISN
3 次元 XYZ 体系	拡散	DIFFUSION / PLOS / CITATION-FBR / DIF3D
	輸送	SNT / TRITAC / PARTISN
3 次元 RθZ 体系	拡散	DIFFUSION
3 次元 Hex-Z 体系	拡散	DIFFUSION / D-MINISTRI / CITATION-FBR / DIF3D
	輸送	MINISTRI7 / NSHEX
3 次元 Tri-Z 体系	拡散	DIFFUSION / D-MINISTRI / CITATION-FBR / DIF3D
	輸送	MINISTRI7

3. ソルバーの開発と改良

3.1 チェビシェフ有理関数近似法に基づく燃焼計算ソルバーの改良

MARBLE2 の燃焼計算ソルバー BURNUP では、チェビシェフ有理関数近似法（CRAM: Chebyshev rational approximation method）[31] に基づく燃焼計算ソルバーを利用することができるが、MARBLE3 では、その後に発表された CRAM の論文 [32,33] に基づいて実装した改良版の CRAM に基づく BURNUP ソルバーについても利用できるように整備した。

この改良版の CRAM に基づく BURNUP ソルバーの一部は MARBLE の機能を利用して開発された燃焼計算コード CRAMO[45] でも利用されている。このため、本章の記載は、CRAMO の報告書 [45] の記載と重複する部分もあるが、MARBLE3 ではオリジナルの CRAM[31]、改良版の CRAM[32,33] を切り替えて利用することができるので、ここでは改めてこれらの解法について説明する。

3.1.1 数値解法

前述のように、MARBLE2 の燃焼計算ソルバーでは CRAM に基づく解法を利用できるようになっている。この MARBLE2 の CRAM に基づく燃焼計算ソルバーのアルゴリズムは、M. Pusa と J. Leppänen の 2010 年の文献 [31] に基づいており、計算に必要な有理関数の係数は、この文献で引用されている 1992 年の E. Gallopoulos と Y. Saad の文献 [46] に掲載されている 10 次と 14 次の係数を使っている。しかしながら、その後も、M. Pusa により燃焼計算における CRAM の応用に関する文献が発表されており、2011 年の文献 [32] には、著者が独自に計算した 14 次と 16 次の係数が掲載されている。また、2012 年の文献 [47] には、1992 年の文献 [46] に掲載されている 14 次の係数を使うと計算精度が悪くなるという指摘もある。更に、2016 年にも文献 [33] が公開されており、この文献では、数値計算誤差をより小さくすることができる IPF（Incomplete Partial Fractions）というアルゴリズムを使うことが提案されており、IPF に対応した 4 次から 48 次までの係数（4、8、12、16、20、24、28、32、36、40、44、48 次の係数）が掲載されている。なお、この論文では、従来の計算式は PFD（Partial Fraction Decomposition）と呼ばれているので、ここでは両手法を、それぞれ、CRAM-IPF、CRAM-PFD と呼んで区別することにする。

MARBLE3 では、新しい文献の情報を参考に、CRAM に基づく燃焼計算ソルバーに対して、以下の 2 点の改良を行った。

- 2011 年の CRAM 論文 [32] に基づく、14 次と 16 次の CRAM-PFD の実装の追加
- 2016 年の CRAM 論文 [33] に基づく、4 次から 48 次までの CRAM-IPF の実装の追加

なお、後述するように、2010 年の論文 [31] のアルゴリズムと 2011 年の論文 [32] のアルゴリズムは同等であり、いずれも CRAM-PFD と呼ぶべきものと考えられるが、MARBLE の開発では、MARBLE2 において 2010 年の論文 [31] の手法と 1992 年の文献 [46] の係数を使った CRAM ソルバーが実装さ

れ、続いて、MARBLE3 において、2011 年の論文 [32] のアルゴリズムと係数を使った CRAM ソルバーが実装された。このため、前者を「CRAM（オリジナル）」、後者を「CRAM-PFD」と呼んで区別することにする。

3.1.1.1 燃焼方程式の数値解法（行列指数法）

CRAM に基づく燃焼計算ソルバーの基本となる行列指数法を用いた燃焼方程式の数値解法について簡単に説明する。一般に、燃焼計算は次に示す燃焼方程式を解くことで行われる。

$$\frac{\mathrm{d}\mathbf{n}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{n}(t) \tag{3.1}$$

ここで、$\mathbf{n}(t)$ は時刻 t における組成ベクトルを、$\mathbf{A}$ は遷移（燃焼）行列を表す。この方程式を解く方法のひとつとして行列指数法がある。行列指数法では燃焼方程式の解を以下のように表すことができる。

$$\mathbf{n}(t) = \exp(\mathbf{A}t)\mathbf{n}_0 \tag{3.2}$$

ただし、ここで $\mathbf{n}_0 = \mathbf{n}(0)$ である。この式から行列指数の $\exp(\mathbf{A}t)$、あるいは、$\exp(\mathbf{A}t)\mathbf{n}_0$ を計算できれば、燃焼後の組成ベクトルを計算できることが分かる。

3.1.1.2 CRAM（オリジナル）による行列指数の数値解法（2010 年 CRAM 論文）

最初に、2010 年の CRAM 論文 [31] に記載されている計算式を確認する。この論文には、スカラーの指数関数を計算する近似式として、以下の式が示されている。

CRAM（オリジナル）のスカラー版【文献記載式】

$$\hat{r}_{k,k}(x) = \alpha_0 + \mathrm{Re}\left(\sum_{i=1}^{k/2} \frac{\alpha_j}{x - \theta_i}\right) \tag{3.3}$$

文献 [31] の (24) 式

この式を行列に拡張し、燃焼計算で必要になる $\mathbf{n}(t)$ を計算する式として以下の式が示されている。

CRAM（オリジナル）のベクトル版（燃焼計算用）【文献記載式】

$$\begin{aligned}\mathbf{n}(t) = e^{\mathbf{A}t}\mathbf{n}_0 &\approx \hat{r}_{k,k}(-\mathbf{A}t)\mathbf{n}_0 \\ &= \alpha_0\mathbf{n}_0 - \mathrm{Re}\left(\sum_{i=1}^{k/2} (\theta_i\mathbf{I} + \mathbf{A}t)^{-1}\,\alpha_i\mathbf{n}_0\right)\end{aligned} \tag{3.4}$$

文献 [31] の (25) 式

しかしながら、この式は、式 (3.3) を行列に拡張した式であるので、逆行列の括弧の中の式の符号は + ではなく − であると推測される。すなわち、正しい式は以下と考えられる。

CRAM（オリジナル）のベクトル版（燃焼計算用）【修正版】

$$\mathbf{n}(t) = \alpha_0 \mathbf{n}_0 + \mathrm{Re}\left(\sum_{i=1}^{k/2} \alpha_i \left(\mathbf{A}t - \theta_i \mathbf{I} \right)^{-1} \mathbf{n}_0 \right) \tag{3.5}$$

3.1.1.3 CRAM-PFD による行列指数の数値解法（2011 年 CRAM 論文）

次に、2011 年の CRAM 論文 [32] に記載されている計算式について確認する。この論文には、スカラーの指数関数を計算する式として、以下の式が示されている。

CRAM-PFD のスカラー版【文献記載式】

$$r_{k,k}(x) = \alpha_0 + 2\mathrm{Re}\left(\sum_{j=1}^{k/2} \frac{\alpha_j}{x - \theta_j} \right) \tag{3.6}$$

文献 [32] の (9) 式

この式は、和を取るときの記号と係数が違うだけで、2010 年の CRAM 論文の計算式（本報告書の式 (3.3)）と同じ形をしていることが分かる。なお、論文には明示されていないが、この式を行列に拡張すると以下のような式になると考えられる。

CRAM-PFD の行列版

$$\hat{r}_{k,k}(\mathbf{A}t) = \alpha_0 + 2\mathrm{Re}\left(\sum_{j=1}^{k/2} \alpha_j \left(\mathbf{A}t - \theta_j \mathbf{I} \right)^{-1} \right) \tag{3.7}$$

更に、燃焼計算用にベクトルを直接計算する場合には、以下のような式になると考えられる。

CRAM-PFD の行列版（燃焼計算用）

$$\begin{aligned} \mathbf{n}(t) &= \exp(\mathbf{A}t)\mathbf{n}_0 \approx \hat{r}_{k,k}(\mathbf{A}t)\mathbf{n}_0 \\ &= \alpha_0 \mathbf{n}_0 + 2\mathrm{Re}\left(\sum_{j=1}^{k/2} \alpha_j \left(\mathbf{A}t - \theta_j \mathbf{I} \right)^{-1} \mathbf{n}_0 \right) \end{aligned} \tag{3.8}$$

実際、論文には以下のような式が記載されており、上記のように拡張して得られた式と考えることができる。

CRAM-PFD の行列版（燃焼計算用）【文献記載式】

$$\mathbf{n} = \alpha_0 \mathbf{n}_0 + 2\mathrm{Re}\left(\sum_{j=1}^{k/2} \alpha_j \left(\mathbf{A}t - \theta_j \mathbf{I} \right)^{-1} \mathbf{n}_0 \right) \tag{3.9}$$

文献 [32] の (10) 式

この式は、2010 年の最初の CRAM 論文の計算式の修正版（本報告書の (3.5) 式）と同じ形をしていることが分かる。また、2016 年の CRAM 論文 [33] にも、CRAM-PFD の式として以下の式が示されているが、この式も同じであることが分かる。

CRAM-PFD の行列版（燃焼計算用）【文献記載式】

$$\hat{r}_{k,k}(\mathbf{A}t)\mathbf{n}_0 = \alpha_0\mathbf{n}_0 + 2\mathrm{Re}\left(\sum_{j=1}^{k/2}\alpha_j\left(\mathbf{A}t-\theta_j\mathbf{I}\right)^{-1}\mathbf{n}_0\right) \tag{3.10}$$

文献 [33] の (17) 式

以上のことから、2010 年のオリジナルの CRAM 論文 [31] の計算方法、2011 年の CRAM 論文 [32] 及び 2016 年の CRAM 論文 [33] で PFD と呼ばれている計算方法は、いずれも同じ計算方法を指していると考えられる。

したがって、今回実装する CRAM-PFD は、これまでの実装と計算方法は同じであり、計算に用いる係数が異なっているだけであると理解できる。

3.1.1.4 CRAM-IPF 法による行列指数の数値解法（2016 年 CRAM 論文）

最後に、2016 年の CRAM 論文 [33] で提案されている CRAM-IPF の計算方法についてまとめる。この論文には、スカラーの指数関数を計算する式として、以下の式が示されている。

CRAM-IPF のスカラー版【文献記載式】

$$\hat{r}_{k,k}(x) = \alpha_0\prod_{l=1}^{k/2}\left(1+2\mathrm{Re}\left(\frac{\widetilde{\alpha}_l}{x-\theta_l}\right)\right) \tag{3.11}$$

文献 [33] の (19) 式

これまでの論文とは異なり、この論文では、CRAM-IPF に基づく燃焼計算用の式は特に示されておらず、文献 [33] には、図 3.1.1 のような計算アルゴリズムのみが示されている（文献 [33] の Algorithm 1）。しかしながら、この計算アルゴリズムを参考に実装したみたところ、正しい計算結果が得られなかった。

このため、これまでの論文と同様に、行列やベクトルの式に拡張して確認する。CRAM-IPF のスカラーの指数関数の式（本報告書の式 (3.11)）を行列に拡張すると以下のような式になると考えられる。

CRAM-IPF の行列版

$$\hat{r}_{k,k}(\mathbf{A}t) = \alpha_0\prod_{l=1}^{k/2}\left(\mathbf{I}+2\mathrm{Re}\left(\alpha_l\left(\mathbf{A}t-\theta_l\mathbf{I}\right)^{-1}\right)\right) \tag{3.12}$$

$\mathbf{e} = [1, 1, \cdots, 1]^T$
$\mathbf{y} = \mathbf{n}(0)$
for $l = 1, 2, \cdots, k/2$ do
  $\mathbf{y} = 2\mathrm{Re}(\alpha_l(\mathbf{A}t - \theta_l\mathbf{I})^{-1}\mathbf{y}) + \mathbf{e}$
end for
$\mathbf{y} = \alpha_0\mathbf{y}$

図 3.1.1　文献記載の CRAM-IPF の計算アルゴリズム

ここで、CRAM-PFD の式を行列に拡張する際にスカラーの θ_j は $\theta_j\mathbf{I}$ に拡張されていることを考慮して、スカラーの 1 は単位行列 $\mathbf{I}$ に拡張した。更に、この式を燃焼計算用のベクトルの式にすると以下のようになると考えられる。

CRAM-IPF のベクトル版（燃焼計算用）

$$\hat{r}_{k,k}(\mathbf{A}t)\mathbf{n}_0 = \alpha_0 \prod_{l=1}^{k/2} \left(\mathbf{n}_0 + 2\mathrm{Re}\left(\alpha_l \left(\mathbf{A}t - \theta_l\mathbf{I}\right)^{-1} \mathbf{n}_0\right)\right) \tag{3.13}$$

ただし、ここで、$\mathbf{In}_0 = \mathbf{n}_0$ となることを用いた。この式の計算アルゴリズムは、図 3.1.2 のようになると考えられる。この図から分かるように、2016 年の CRAM 論文 [33] の Algorithm 1 のループの中の式の最後の項の $\mathbf{e}$ は、$\mathbf{y}$ のタイプミスと思われる。この場合、$\mathbf{e}$ の定義は必要なくなる。実際に、この計算アルゴリズムで実装したところ、他の数値解法と等価な結果が得られるようになったため、MARBLE の CRAM-IPF に基づく燃焼計算ソルバーの実装では、図 3.1.2 に示した計算アルゴリズムを採用した。

$\mathbf{y} = \mathbf{n}(0)$
for $l = 1, 2, \cdots, k/2$ do
  $\mathbf{y} = 2\mathrm{Re}(\alpha_l(\mathbf{A}t - \theta_l\mathbf{I})^{-1}\mathbf{y}) + \mathbf{y}$
end for
$\mathbf{y} = \alpha_0\mathbf{y}$

図 3.1.2　CRAM-IPF の計算アルゴリズム（修正版）

3.1.1.5 補足：逆行列とベクトルの積の数値計算方法

CRAM-IPF の計算アルゴリズムでは、$(\mathbf{A}t - \theta_l \mathbf{I})^{-1}\,\mathbf{y}$ のような逆行列とベクトルの積を計算する必要がある。同様の計算は、オリジナルの CRAM や CRAM-PFD の計算アルゴリズムでも必要になるが、MARBLE2 の報告書 [8] でも説明しているように、この逆行列を計算してからベクトルとの積を計算するのではなく、MARBLE2 の CRAM ソルバーの実装では、線形方程式の解として最終的に必要となるベクトルを直接計算することで高速化している。このため、CRAM-PFD、CRAM-IPF の実装においてもこの方法を用いて高速化を行った。

なお、この逆行列とベクトルの積が線形方程式の解として計算できることは次のようにして確認できる。CRAM-IPF の計算アルゴリズムで計算する必要のある逆行列とベクトルの積を以下のようにおく。

$$\mathbf{x} = (\mathbf{A}t - \theta_l \mathbf{I})^{-1}\,\mathbf{y} \tag{3.14}$$

このとき、$\mathbf{x}$ は以下の線形方程式の解として計算することができる。

$$(\mathbf{A}t - \theta_l \mathbf{I})\,\mathbf{x} = \mathbf{y} \tag{3.15}$$

3.1.2 実装

燃焼計算に必要な燃焼チェーンや初期組成の入力設定等の機能は、既に MARBLE の機能として実装されているので、MARBLE3 では、基本的に CRAM に基づく行列指数の計算機能のみを追加した。MARBLE3 では、MARBLE2 でも利用可能であった CRAM（オリジナル）に基づく行列指数計算モジュール CramMatrixExponential に加えて、CRAM-PFD、CRAM-IPF に基づく行列指数計算モジュールとして、それぞれ、CramPfdMatrixExponential モジュール、CramIpfMatrixExponential モジュールが追加されている。これらのモジュールには、以下に示すような関数が含まれている。

- marble.solvercore.burnup.CramPfdMatrixExponential
  - get_pfd_coefficients 関数（CRAM-PFD の係数の定義）
  - exp 関数（スカラー版）
  - expm 関数（行列版（逆行列計算））
  - expv_ver0 関数（ベクトル版（逆行列計算））
  - expv 関数（ベクトル版（線形方程式計算））

- marble.solvercore.burnup.CramIpfMatrixExponential
  - get_ipf_coefficients 関数（CRAM-IPF の係数の定義）
  - exp 関数（スカラー版）
  - expm 関数（行列版（逆行列計算）、(3.12) 式の直訳的実装）
  - expv_ver0 関数（ベクトル版（逆行列計算）、(3.13) 式の直訳的実装）

- expv_algorithm1 関数（ベクトル版（逆行列計算）、図 3.1.2 の修正版アルゴリズム）
- expv 関数（ベクトル版（線形方程式計算）、図 3.1.2 の修正版アルゴリズム）

実用上は、最も計算効率の良い expv 関数のみがあればよく、実際、SCHEME や ORPHEUS の BURNUP ソルバーとしては expv 関数を使用しているが、アルゴリズムの確認等に利用したスカラー版の指数関数や他のアルゴリズムに基づく行列指数関数の実装も含まれている。

3.1.3 検証

MARBLE3 で追加された CRAM-IPF に基づく燃焼計算ソルバーの実装については、フランスの原子力・代替エネルギー庁（CEA）で開発されている MENDEL コードシステム [48] とのベンチマークを実施している [49]。このベンチマークでは、Pu-239 の瞬時照射に対する崩壊熱計算のベンチマークと高速炉を対象とした燃焼計算のベンチマークが行われた。

崩壊熱計算のベンチマークでは、MARBLE の CRAM-PFD ソルバーが用いられたが、MARBLE と MENDEL の結果はよく一致することが確認された。冷却期間を通して、β線による発熱、γ線による発熱、発熱の合計のいずれで比較しても、0.01%以下の差で一致することが示されている。

燃焼計算のベンチマークでは、MARBLE、MENDEL の双方で独立に実装された CRAM-IPF ソルバーを使ったベンチマークが行われた。このベンチマークでは、JENDL-3.3 に基づく ORIGEN2 ライブラリ（ORLIBJ33）[50] を MARBLE、MENDEL のそれぞれで読込み、典型的な高速炉の酸化物燃料組成を 375 日間照射した後の原子数密度の計算結果の比較を行った。このベンチマークでは MARBLE、MENDEL ともに 48 次の CRAM-IPM ソルバーが用いられた。このベンチマークでは、Pd-102、Cd-108、Cd-109、Sn-114、Ba-132 の 5 核種で数%から 10%の差が見られたものの、ORIGEN2 で計算対象としている他の全ての核種については、0.2%以下の差で一致したことが示されている。

3.2 ミニマックス多項式近似法に基づく燃焼計算ソルバーの整備

前述のチェビシェフ有理関数近似法に加えて、2015 年にもう一つ別の新しい燃焼計算手法として、ミニマックス多項式近似（MMPA: Mini-Max Polynomial Approximation）法が提案された [34]。MARBLE3 では、この MMPA 法に基づく燃焼計算ソルバーも整備した。

3.2.1 数値解法

3.2.1.1 燃焼方程式の数値解法（行列指数法）

MMPA 法は CRAM と同様に行列指数法を用いた燃焼方程式の数値解法である。CRAM の数値解法の説明の際に示した燃焼方程式と行列指数法による解の式を再掲する。前述のように、行列指数法では、燃焼方程式

$$\frac{\mathrm{d}\mathbf{n}(t)}{\mathrm{d}t} = \mathbf{A}\mathbf{n}(t) \tag{3.16}$$

に対する解を以下のように表すことができる。

$$\mathbf{n}(t) = e^{\mathbf{A}t}\mathbf{n}_0 \tag{3.17}$$

3.2.1.2 MMPA 法による行列指数の数値解法

MMPA 法を用いると、スカラー値 x の指数関数に対して以下の近似式が成り立つ [34]。

$$e^x \approx f(x) = a_0 + \sum_{i=1}^{n} a_i \left(\frac{x+c}{x-c}\right)^i \tag{3.18}$$

文献 [34] の (14) 式

この式を行列の式に拡張すると以下の式が得られる。

$$e^{\mathbf{A}t} \approx a_0\mathbf{I} + \sum_{i=1}^{n} a_i \left((\mathbf{A}t + c\mathbf{I})\,(\mathbf{A}t - c\mathbf{I})^{-1}\right)^i \tag{3.19}$$

文献 [34] の (4) 式

ただし、ここで、a_i は近似係数、n は展開次数、c は任意の定数である。

なお、32 次のときは、任意の定数 $c = 24.1$ のときに (3.19) 式の誤差が最小になる [34] ことが示されており、文献 [34] の Table 1 に、このときの近似係数 $a_0, a_1, \cdots, a_{32}$ が示されている。

MARBLE3 ではこの文献のアルゴリズムと係数を使って MMPA 法に基づく燃焼計算ソルバーを実装した。

3.2.1.3 多項式の計算アルゴリズム

基本的には、式 (3.19) に基づいて実装すればよいが、(3.19) 式に含まれる多項式を式のとおりに単純に計算すると効率が悪くなるので、数値計算上の工夫が必要となる。

(1)　単純な方法

例えば、以下のような多項式

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n \tag{3.20}$$

を式のとおりに計算すると

$$0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \tag{3.21}$$

回の乗算が必要になる。

この式の計算アルゴリズムを Python を使った疑似コードで書くと図 3.2.1 のようになる。燃焼計算ソルバーの実装では、x は行列になるので、行列のべき乗を繰り返すことになってしまい、この計算アルゴリズムでは非常に計算効率が悪くなってしまう。このため、ここでは、多項式の効率的な計算方法としてよく知られているホーナー法を使って実装した。

```
px = 0
for i, ai in enumerate(a):
    px += ai * x**i
```

図 3.2.1　単純な多項式の計算アルゴリズム

(2)　ホーナー法（行列-行列積）

ホーナー法では、式 (3.20) で表される多項式を以下のように変形して計算する。

$$p_n(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_i + \cdots + x(a_{n-1} + x(a_n)) \cdots) \cdots)) \tag{3.22}$$

この式を用いると、n 回の乗算で計算することができる。しかしながら、このように一般的に n 次の場合について式を書くと繰り返し計算の関係が分かりにくいので、次数を固定して 4 次の場合で説明する。この場合の多項式（4 次多項式）は以下のように変形できる。

$$\begin{aligned} p_4(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \\ &= a_0 + a_1 x + a_2 x^2 + x(a_3 x^2 + a_4 x^3) \\ &= a_0 + a_1 x + x(a_2 x + x(a_3 x + a_4 x^2)) \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + a_4 x))) \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + x(a_4)))) \end{aligned} \tag{3.23}$$

この式において、括弧の内側から順番に計算していくと考えると、4 次多項式は 4 回の乗算（この式からの類推として n 次多項式は n 回の乗算）で計算できることが分かる。

この計算アルゴリズムを Python を使った疑似コードで書くと図 3.2.2 のようになる。スカラーの多項式の場合は、この計算アルゴリズムで十分であるが、前述のように、ここでは、行列の多項式を計算しなければならないので、px と x は行列となり、このアルゴリズムを使うと n 回の行列と行列の積の計算が必要になる。

```
px = a[-1]
for ai in reversed(a[:-1]):
    px = x * px + ai
```

図 3.2.2　ホーナー法による多項式の計算アルゴリズム（行列-行列積）

一方で、前述のように、燃焼方程式を解くという目的においては、必ずしも、$\exp(\mathbf{A}t)$ で表される行列を求める必要はなく、$\exp(\mathbf{A}t)\mathbf{n}_0$ で表されるベクトルが得られれば良い。このことを利用すると、更に計算量を減らすことができる。

(3)　ホーナー方法（行列-ベクトル積）

燃焼方程式の解としては、行列 $p_n(x)$ に初期値のベクトル v をかけたベクトルが得られれば十分である。したがって、以下のような式変形が可能であることを考慮すると、n 回の行列とベクトルの積の計算で済むことが分かる。

$$\begin{aligned} p_4(x)v &= (a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4)v \\ &= a_0v + a_1xv + a_2x^2v + a_3x^3v + a_4x^4v \\ &= a_0v + a_1xv + a_2x^2v + x(a_3x^2v + a_4x^3v) \\ &= a_0v + a_1xv + x(a_2xv + x(a_3xv + a_4x^2v)) \\ &= a_0v + x(a_1v + x(a_2v + x(a_3v + a_4xv))) \\ &= a_0v + x(a_1v + x(a_2v + x(a_3v + x(a_4v)))) \end{aligned} \tag{3.24}$$

この計算アルゴリズムを Python を用いた疑似コードで書くと図 3.2.3 のようになる。この場合、x は行列、pxv はベクトル、ai はスカラーになるので、n 回の行列とベクトルの積と、$(n+1)$ 回のベクトルとスカラーの積で計算できることが分かる。

```
pxv = a[-1] * v
for ai in reversed(a[:-1]):
    pxv = x * pxv + ai * v
```

図 3.2.3　ホーナー法による多項式の計算アルゴリズム（行列-ベクトル積）

3.2.2 実装

CRAM に基づく燃焼計算ソルバーの実装と同様に、燃焼計算に必要な燃焼チェーンや初期組成の入力設定等の機能は、既に MARBLE の機能として実装されているので、MARBLE3 では、基本的に MMPA 法に基づく行列指数の計算機能のみ追加した。具体的には、MARBLE3 では、MMPA 法に基づく行列指数を計算する MmpaMatrixExponential モジュールを実装した。MmpaMatrixExponential モジュールには、以下に示すような関数が含まれている。

- marble.solvercore.burnup.MmpaMatrixExponential
  - get_mmpa_coefficients 関数（MMPA の近似係数の定義）
  - exp 関数（MMPA 法に基づくスカラー指数計算）
  - expm_ver0 関数（MMPA 法に基づく行列指数計算（単純な多項式の計算法））
  - expm 関数（MMPA 法に基づく行列指数計算（ホーナー法：行列-行列積））
  - expv 関数（MMPA 法に基づく行列指数計算（ホーナー法：行列-ベクトル積））

CRAM の実装と同様に、実用上は、最も計算効率の良い expv 関数のみがあればよいが、アルゴリズムの確認等に利用したスカラー版の指数関数や他のアルゴリズムに基づく行列指数関数の実装も含まれている。

3.2.3 検証

MmpaMatrixExponential モジュールに含まれる関数に対する単体テストでは、他の行列指数関数のモジュールと同様に、乱数で作成した 3×3 行列に対して行列指数計算を行い、SciPy が提供するパーデ近似に基づく行列指数の関数（scipy.linalg.expm）と等価な結果が得られることを確認している。また、MMPA 法に基づく燃焼計算ソルバーの単体テストでは、他の解法に基づく燃焼計算ソルバーと同様に、MARBLE が提供する重核種のみの燃焼チェーン（standard2006）を使った計算において、テイラー展開法に基づく燃焼計算ソルバーと等価な結果が得られることを確認している。

MMPA 法に基づく燃焼計算ソルバーについては、CRAM に基づく燃焼計算ソルバーのような他のコードシステムとのベンチマークは実施できていないので、今後、より規模の大きい燃焼チェーンを用いた検証計算が必要と考えられる。

3.3 一点炉動特性ソルバーの再実装

MARBLE2 には、一点炉動特性方程式を解くためのソルバーとして Pointkinetics モジュール [15] が組み込まれていた。しかしながら、前述のように、Pointkinetics モジュールの実装では、現在、開発が実質上停止している C++の数値計算ライブラリを利用していたため、MARBLE3 では、インストールの煩雑さとソースコードの維持管理の煩雑さを緩和するために、Python の数値計算ライブラリを使って再実装し、同等の機能を持つ Kinetics モジュールとして整備し直した。また、Blitz++は標準的な Linux のディストリビューションにおいてパッケージとして提供されていないことも多く、MARBLE をインストールを難しくする原因の一つとなっていた。このため、Python の標準的な数値計算ライブラリとして広く利用されており、インストールが容易で扱いやすく、MARBLE でも汎用的な数値計算ライブラリとして利用している NumPy[23] と SciPy[24] を使って再実装した。

ここでは、一点炉動特性ソルバー KINETICS（Kinetics モジュール）の実装について説明する。Pointkinetics の報告書 [15] には、Pointkinetics モジュールの実装に用いた一点炉動特性方程式の数値解法が詳細に記載されている。Kinetics モジュールもこの報告書の記述に基づいて再実装したので、Pointkinetics の報告書の説明があれば、今回再実装した Kinetics モジュールの実装についても理解できると考えられる。しかしながら、少し補足説明があると理解しやすい部分もあるので、Pointkinetics の報告書と繰り返しになる部分も多いが、Kinetics モジュールの実装に用いた式を再掲して、Kinetics モジュールの実装について説明する。ただし、Pointkinetics の報告書の式から一部の記号を変更している。また、変数を明示する等の微修正を行っている部分もある。

3.3.1 数値解法

3.3.1.1 一点炉動特性方程式

一点炉動特性方程式は以下のように表すことができる。

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \frac{\rho(t) - \beta}{\Lambda} n(t) + \sum_i \lambda_i C_i(t) + S \tag{3.25}$$

$$\frac{\mathrm{d}C_i}{\mathrm{d}t} = \frac{\beta_i}{\Lambda} n(t) - \lambda_i C_i(t) \tag{3.26}$$

Pointkinetics の報告書の (2-16) – (2-17) 式

ただし、ここで、

$n(t)$ ：時刻 t における中性子の数密度
$C_i(t)$ ：時刻 t における第 i 群の遅発中性子先行核の濃度
S ：外部中性子源
$\rho(t)$ ：時刻 t における反応度
β_i ：第 i 群の遅発中性子先行核の遅発中性子割合

β ：遅発中性子割合の合計（$\beta \equiv \sum_i \beta_i$）
λ_i ：第 i 群の遅発中性子先行核の崩壊定数
Λ ：中性子生成時間

である。

3.3.1.2　一点炉動特性方程式の参照解

最初に、Pointkinetics モジュールで実装されている参照解の式についてまとめる。

(1)　ステップ入力に対する厳密解

ステップ状の反応度 ρ が投入されたときの時刻 t における中性子数密度は、解析的に求めることが可能であり、以下の式で表される [15]。

$$n(t) = n_0 \sum_j A_j \exp(\omega_j t) + S \sum_j \frac{B_j}{\omega_j} \left(\exp(\omega_j t) - 1\right) \tag{3.27}$$

Pointkinetics の報告書の (2-28) 式

ただし、ここで、n_0 は時刻 $t = 0$ における中性子数密度であり、

$$A_j = \frac{\Lambda + \sum_i \dfrac{\beta_i}{\omega_i + \lambda_i}}{\Lambda + \sum_i \dfrac{\beta_i \lambda_i}{(\omega_i + \lambda_i)^2}} \tag{3.28}$$

$$B_j = \frac{\Lambda}{\Lambda + \sum_i \dfrac{\beta_i \lambda_i}{(\omega_i + \lambda_i)^2}} \tag{3.29}$$

Pointkinetics の報告書の (2-26) – (2-27) 式

である。また、ω_i は、以下の特性方程式（逆時間方程式）の解を表す。

$$\Lambda\omega + \beta - \rho - \sum_i \frac{\beta_i \lambda_i}{\omega + \lambda_i} = 0 \tag{3.30}$$

Pointkinetics の報告書の (2-24) 式

Pointkinetics の報告書に記載されているように、Pointkinetics モジュールでは、逆時間方程式をニュートン法により解いている。この部分を Python で実装すると計算に時間がかかるため、Pointkinetics モジュールでは C++（Blitz++）を使って実装している。一方、今回再実装した Kinetics モジュールでは、逆時間方程式を Scipy のライブラリで提供されている Brent 法に基づいて最小値を求める関数（scipy.optimize.brentq）を用いて逆時間方程式を解くようにした。なお、scipy.optimize ではニュートン法の関数（scipy.optimize.newton）や他の解法に基づく関数も提供されている。この

ため、Pointkinetics と同様にニュートン法の関数を使った実装も試してみたが、安定して解を見つけられないことがあったため、Kinetics モジュールでは Brent 法に基づく関数を利用することにした。

(2) 線形近似解

ある定常状態（中性子の数密度、遅発中性子先行核の濃度、反応度を、それぞれ、n_0、C_{i0}、ρ_0とする）仮定し、反応度の変化が発生した際の中性子数密度、遅発中性子先行核濃度の変化を以下のように表す。

$$\rho = \rho_0 + \delta\rho \tag{3.31}$$

$$n = n_0 + \delta n \tag{3.32}$$

$$C_i = C_{i0} + \delta C_i \tag{3.33}$$

Pointkinetics 報告書の (2-29) – (2-30) 式

このとき、線形近似を使うと反応度のランプ入力や正弦波入力に対する解を解析的に求めることができる [15]。

(3) 線形近似解（ランプ入力）

以下のような反応度のランプ入力に対して、

$$\delta\rho(t) = at \tag{3.34}$$

Pointkinetics 報告書の (2-44) 式

時刻 t における反応度変化は以下の式で表される。

$$\delta n(t) = n_0 \sum_i a A_j \left(\frac{1}{\omega_j^2} \left(\exp(\omega_j t) - 1 \right) - \frac{t}{\omega_j} \right) \tag{3.35}$$

Pointkinetics 報告書の (2-47) 式

ただし、ここで、

$$A_j = \frac{1}{\Lambda + \sum_i \dfrac{\beta_i \lambda_i}{(\omega_j + \lambda_i)^2}} \tag{3.36}$$

Pointkinetics 報告書の (2-43) 式

である。

(4)　線形近似解（正弦波入力）

以下のような反応度の正弦波入力に対して、

$$\delta\rho(t) = a \sin\left(\frac{2\pi}{T}t\right) \tag{3.37}$$

Pointkinetics 報告書の (2-48) 式

時刻 t における反応度変化は以下の式で表される。

$$\delta n(t) = n_0 \sum_j \frac{a\frac{2\pi}{T}A_j}{\omega_j^2 + \left(\frac{2\pi}{T}\right)^2}\left(\exp(\omega_j t) - \frac{\omega_j}{\frac{2\pi}{T}}\sin\left(\frac{2\pi}{T}t\right) - \cos\left(\frac{2\pi}{T}t\right)\right) \tag{3.38}$$

Pointkinetics 報告書の (2-51) 式

ただし、ここで、A_j は式 (3.36) で表される。

3.3.1.3　一点炉動特性方程式の数値解法

続いて、Pointkinetics モジュールで実装されている数値解法の式についてまとめる。

(1)　行列とベクトルを用いた一般的な表式

Pointkinetics の報告書の第 3 章の冒頭で説明されているように、一点炉動特性方程式の解法は、特殊な手法を除けば、連立一階微分方程式の一般的な解法として議論することができる。

遅発中性子先行核が N 群の場合について、一点炉動特性方程式を具体的に遅発中性子先行核 $i = 1, 2, \cdots, N$ について書くと、以下のような $(N+1)$ 元連立一階微分方程式になる。

$$\begin{cases} \dfrac{\mathrm{d}n}{\mathrm{d}t} = \dfrac{\rho-\beta}{\Lambda}n + \lambda_1 C_1 + \lambda_2 C_2 + \cdots + \lambda_N C_N + S \\ \dfrac{\mathrm{d}C_1}{\mathrm{d}t} = \dfrac{\beta_1}{\Lambda}n - \lambda_1 C_1 \\ \dfrac{\mathrm{d}C_2}{\mathrm{d}t} = \dfrac{\beta_2}{\Lambda}n - \lambda_2 C_2 \\ \qquad \vdots \\ \dfrac{\mathrm{d}C_N}{\mathrm{d}t} = \dfrac{\beta_N}{\Lambda}n - \lambda_N C_N \end{cases} \tag{3.39}$$

したがって、一点炉動特性方程式は、以下のように行列とベクトルを使って表すことができる。

$$\begin{pmatrix} \frac{\mathrm{d}n}{\mathrm{d}t} \\ \frac{\mathrm{d}C_1}{\mathrm{d}t} \\ \frac{\mathrm{d}C_2}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}C_N}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} \frac{\rho-\beta}{\Lambda}n & \lambda_1 & \lambda_2 & \cdots & \lambda_N \\ \frac{\beta_1}{\Lambda} & -\lambda_1 & 0 & \cdots & 0 \\ \frac{\beta_2}{\Lambda} & 0 & -\lambda_2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \frac{\beta_N}{\Lambda} & 0 & \cdots & 0 & -\lambda_N \end{pmatrix} \begin{pmatrix} n \\ C_1 \\ C_2 \\ \vdots \\ C_N \end{pmatrix} + \begin{pmatrix} S \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{3.40}$$

すなわち、一点炉動特性方程式は、行列 **A**、ベクトル $\boldsymbol{x}$、$\boldsymbol{b}$ を使って、以下のような一般的な形式で表すことができる。

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \mathbf{A}\boldsymbol{x} + \boldsymbol{b} \tag{3.41}$$

Pointkinetics 報告書の (3-1) 式

なお、本報告書では、上式のように、行列を大文字のゴシック体、ベクトルを小文字の斜体ゴシック体で表すことにする。

(2) 行列指数法

行列指数法を用いた場合、時刻 t_{n+1} における解は一つ前の時刻 t_n における解を使って以下のように計算することができる [15)]。

$$\boldsymbol{x}(t_{n+1}) = -\mathbf{A}^{-1}\boldsymbol{b} + \exp(\Delta t\mathbf{A})(\boldsymbol{x}(t_n) + \mathbf{A}^{-1}\boldsymbol{b}) \tag{3.42}$$

Pointkinetics 報告書の (3-8) 式

指数行列の計算には、Scipy の関数（scipy.linalg.expm）を使うことができる。この関数は Padé 近似で行列指数を計算するものであり、Pointkinetics モジュールの実装でも用いられていた。また、この方法では **A** の逆行列を計算する必要があるが、**A** は、例えば、遅発中性子先行核が 6 群の場合、7×7 の行列でありそれほど大きな行列ではないので、NumPy の関数（numpy.linalg.inv）を利用した。

なお、この式の導出については、Pointkinetics の報告書でも説明されているが、この行列指数法が Kinetics モジュールにおける実質上の標準解法となるので、この式の導出の詳細を以下にまとめる。

(3) 行列指数法の式の導出の詳細

Pointkinetics の報告書の第 3 章に示されているように、$\boldsymbol{x}'(t)$ を以下のように定義する。

$$\boldsymbol{x}'(t) \equiv \boldsymbol{x}(t) + \mathbf{A}^{-1}\boldsymbol{b} \tag{3.43}$$

Pointkinetics 報告書の (3-9) 式

このとき、一点炉動特性方程式は以下のように表される。

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{x}'(t) - \mathbf{A}^{-1}\boldsymbol{b}\right) &= \mathbf{A}\left(\boldsymbol{x}'(t) - \mathbf{A}^{-1}\boldsymbol{b}\right) + \boldsymbol{b} \\
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}'(t) &= \mathbf{A}\boldsymbol{x}'(t) - \mathbf{A}\mathbf{A}^{-1}\boldsymbol{b} + \boldsymbol{b} \\
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{x}'(t) &= \mathbf{A}\boldsymbol{x}'(t)
\end{aligned} \tag{3.44}$$

この微分方程式の解は、行列指数を用いて以下のように表すことができる。

$$\boldsymbol{x}'(t_{n+1}) = \exp(\Delta t\mathbf{A})\boldsymbol{x}'(t_n) \tag{3.45}$$

この式に式 (3.43) を代入すると以下の式が得られる。

$$\begin{aligned}
\boldsymbol{x}(t_{n+1}) + \mathbf{A}^{-1}\boldsymbol{b} &= \exp(\Delta t\mathbf{A})\left(\boldsymbol{x}(t_n) + \mathbf{A}^{-1}\boldsymbol{b}\right) \\
\boldsymbol{x}(t_{n+1}) &= -\mathbf{A}^{-1}\boldsymbol{b} + \exp(\Delta t\mathbf{A})\left(\boldsymbol{x}(t_n) + \mathbf{A}^{-1}\boldsymbol{b}\right)
\end{aligned} \tag{3.46}$$

この式は、式 (3.42) と同じであることが確認できる。

(4) 陽解法

陽解法を用いた場合、以下のように計算することができる [15)]。

$$\boldsymbol{x}(t_{n+1}) = \boldsymbol{x}(t_n) + \Delta t\left(\mathbf{A}\boldsymbol{x}(t_n) + \boldsymbol{b}\right) \tag{3.47}$$

Pointkinetics 報告書の (3-16) 式

(5) 陰解法

陰解法を用いた場合、以下のように計算することができる [15)]。

$$\boldsymbol{x}(t_{n+1}) = \left(\mathbf{I} - \Delta t\mathbf{A}\right)^{-1}\left(\boldsymbol{x}(t_n) + \Delta t\boldsymbol{b}\right) \tag{3.48}$$

Pointkinetics 報告書の (3-17) 式

(6) θ 法

θ 法を用いた場合、以下のように計算することができる [15)]。

$$\boldsymbol{x}(t_{n+1}) = \left(\mathbf{I} - \theta\Delta t\mathbf{A}\right)^{-1}\left(\left(\mathbf{I} + (1-\theta)\Delta t\mathbf{A}\right)\boldsymbol{x}(t_n) + \Delta t\boldsymbol{b}\right) \tag{3.49}$$

Pointkinetics 報告書の (3-19) 式

(7)　ルンゲクッタ法

Pointkinetics の報告書では、スカラーの x に対するルンゲクッタの法の説明がされているが、NumPy の配列を使ってベクトルの $\boldsymbol{x}$ に対して直接実装することができるので、ここでは、他の解法と同様にベクトルの $\boldsymbol{x}$ を使って実装の式を整理する。Pointkinetics モジュールでは 4 次のルンゲクッタ法を用いていたので、Kinetics モジュールでも同じ方法を用いた。

以下の微分方程式に対する解は、

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = f(t, \boldsymbol{x}) \tag{3.50}$$

Pointkinetics 報告書の (3-23) 式

4 次のルンゲクッタ法を用いると以下のように表すことができる。

$$\boldsymbol{x}(t_{n+1}) = \boldsymbol{x}(t_n) + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{3.51}$$

Pointkinetics 報告書の (3-28) 式

ただし、ここで、

$$k_1 = f(t_n, \boldsymbol{x}(t_n)) \tag{3.52}$$

$$k_2 = f\left(t_n + \frac{\Delta t}{2}, \boldsymbol{x}(t_n) + \frac{\Delta t}{2}k_1\right) \tag{3.53}$$

$$k_3 = f\left(t_n + \frac{\Delta t}{2}, \boldsymbol{x}(t_n) + \frac{\Delta t}{2}k_2\right) \tag{3.54}$$

$$k_4 = f\left(t_n + \Delta t, \boldsymbol{x}(t_n) + \Delta t k_3\right) \tag{3.55}$$

Pointkinetics 報告書の (3-24) – (3-27) 式

である。

(8)　時間積分法

Pointkinetics の報告書に記載されているように、時間積分法は一点炉動特性方程式に特化した解法であり、時刻 t_{n+1} における中性子数密度、遅発中性子先行核濃度は、それぞれ、以下の式で計算することができる [15]。

$$n(t_{n+1}) = \frac{\left(1 + \Delta t \sum_i \frac{\beta_i}{\Lambda}\left(\frac{1-\exp(-\lambda_i \Delta t)}{\lambda_i \Delta t} - \exp(-\lambda_i \Delta t)\right)\right) n(t_n) + \Delta t \sum_i \lambda_i C_i(t_n) \exp(-\lambda_i \Delta t) + S\Delta t}{1 - \Delta t \frac{\rho - \beta}{\Lambda} - \Delta t \sum_i \frac{\beta_i}{\Lambda}\left(1 - \frac{1-\exp(-\lambda_i \Delta t)}{\lambda_i \Delta t}\right)} \tag{3.56}$$

Pointkinetics 報告書の (3-32) 式

$$C_i(t_{n+1}) = C_i(t_n)\exp(-\lambda_i\Delta t) + \frac{\beta_i}{\lambda_i\Lambda}\left(\left(\frac{1-\exp(-\lambda_i\Delta t)}{\lambda_i\Delta t} - \exp(-\lambda_i\Delta t)\right)n(t_n) + \left(1 - \frac{1-\exp(-\lambda_i\Delta t)}{\lambda_i\Delta t}\right)n(t_{n+1})\right) \quad (3.57)$$

Pointkinetics 報告書の (3-31) 式

(9)　SCM（Stiffness Confinement Method）

SCM については、Pointkinetics モジュールでも実装されていなかったので、Kinetics モジュールでも実装していない。

3.3.1.4　一点炉動特性方程式の数値解法（即発跳躍近似）

Pointkinetics モジュールでは、陽解法、陰解法、θ 法、ルンゲクッタ法、時間積分法に対して、即発跳躍近似を適用したソルバーも提供されている。Kinetics モジュールでも同様にこれらの解法に対して即発跳躍近似を適用したソルバーを再実装した。

(1)　即発跳躍近似（陽解法、陰解法、θ 法、ルンゲクッタ法）

Pointkinetics の報告書の式 (2-16) に対応する本報告書の式 (3.25) において、時間変化項を無視する（$\frac{\mathrm{d}n}{\mathrm{d}t} = 0$）と、式 (3.25) は、以下のように変形できる。

$$\begin{aligned} 0 &= \frac{\rho(t)-\beta}{\Lambda}n(t) + \sum_i \lambda_i C_i(t) + S \\ -\frac{\rho(t)-\beta}{\Lambda}n(t) &= \sum_i \lambda_i C_i(t) + S \\ n(t) &= \frac{\Lambda}{\beta-\rho(t)}\left(\sum_i \lambda_i C_i(t) + S\right) \end{aligned} \quad (3.58)$$

このように、即発跳躍近似を適用した場合の時刻 t における中性子数密度 $n(t)$ は、式 (3.58) を使って直接計算できるので、式 (3.39) と同様に一点炉動特性方程式を書くと以下のような遅発中性子先行核濃度のみの N 元連立一階微分方程式になる。

$$\begin{cases} \dfrac{\mathrm{d}C_1}{\mathrm{d}t} = \dfrac{\beta_1}{\Lambda}n(t) - \lambda_1 C_1 \\ \dfrac{\mathrm{d}C_2}{\mathrm{d}t} = \dfrac{\beta_2}{\Lambda}n(t) - \lambda_2 C_2 \\ \quad\vdots \\ \dfrac{\mathrm{d}C_N}{\mathrm{d}t} = \dfrac{\beta_N}{\Lambda}n(t) - \lambda_N C_N \end{cases} \quad (3.59)$$

したがって、即発跳躍近似を適用した場合の一点炉動特性方程式は、以下のように行列とベクトルを使って表すことができる。

$$\begin{pmatrix} \frac{\mathrm{d}C_1}{\mathrm{d}t} \\ \frac{\mathrm{d}C_2}{\mathrm{d}t} \\ \vdots \\ \frac{\mathrm{d}C_N}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} -\lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_N \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{pmatrix} + \begin{pmatrix} \frac{\beta_1}{\Lambda} n(t) \\ \frac{\beta_2}{\Lambda} n(t) \\ \vdots \\ \frac{\beta_N}{\Lambda} n(t) \end{pmatrix} \tag{3.60}$$

ただし、ここで、$n(t)$ は式 (3.58) で表される。

Pointkinetics モジュールにおける陽解法、陰解法、θ 法、ルンゲクッタ法の即発跳躍近似のソルバーにおいて上記のような計算が行われていることを確認し、Kinetics モジュールでも同様の計算方法を再実装した。

(2)　即発跳躍近似（時間積分法）

時間積分法の即発跳躍近似のソルバーに関しては、Pointkinetics の報告書 [15] に記載されている以下の式を用いて再実装した。

$$n(t_{n+1}) = \frac{\Lambda \left(\sum_i \lambda_i C_i(t_n) + S \right)}{\beta - \rho} \tag{3.61}$$

Pointkinetics 報告書の (3-35) 式

$$C_i(t_{n+1}) = C_i(t_n) \exp(-\lambda_i \Delta t) + \frac{\beta_i \left(1 - \exp(-\lambda_i \Delta t)\right) n(t_n)}{\lambda_i \Lambda} \tag{3.62}$$

Pointkinetics 報告書の (3-34) 式

3.3.1.5　一点炉動特性方程式の逆解法

Pointkinetics モジュールでは、中性子数密度（原子炉出力）の変化から反応度の変化を逆算する逆解法のソルバーも実装されているので、Kinetics モジュールでも同様に逆解法のソルバーを再実装した。

時刻 t_n における反応度 $\rho(t_n)$ は、時刻 t_n、t_{n-1} における中性子数密度 $n(t_n)$、$n(t_{n-1})$ と、時刻 t_{n-1} における遅発中性子先行核濃度 $C_i(t_{n-1})$ を用いて、以下の式で計算することができる [15]。

$$\rho(t_n) = \beta + \Lambda \omega_n - \frac{\Lambda \sum_i \lambda_i C_i(t_n)}{n(t_n)} - \frac{\Lambda S}{n(t_n)} \tag{3.63}$$

Pointkinetics 報告書の (4-5) 式

ただし、ここで、

$$\omega_n = \frac{\ln\left(n(t_n)\right) - \ln\left(n(t_{n-1})\right)}{\Delta t} \tag{3.64}$$

Pointkinetics 報告書の (4-6) 式

$$C_i(t_n) = C_i(t_{n-1})\exp(-\lambda_i \Delta t) + \frac{\beta_i}{\Lambda} \frac{\dfrac{n(t_n)}{\omega_n} - \dfrac{n(t_{n-1})}{\omega_n}\exp(-\lambda_i \Delta t)}{1 + \dfrac{\lambda_i}{\omega_n}} \tag{3.65}$$

Pointkinetics 報告書の (4-8) 式

である。

3.3.2 実装

今回再実装した Kinetics モジュール（marble.solvercore.kinetics パッケージ）に含まれる主なモジュールとクラス、関数についてまとめる。Kinetics モジュールでは、現在の MARBLE の命名規則に合わせてクラス名等を少し修正しているが、機能を提供するクラスや関数の粒度は、Pointkinetics モジュールとほとんど同じである。一方で、参照解のソルバーや逆解法のソルバーは、通常の一点炉動特性ソルバーとは少し役割が異なるので、別のモジュールに分割した。このように Kinetics モジュールでは名前や構造が少し変更されているが、提供する機能については大きな変更はないので、Pointkinetics モジュールのユーザは容易に Kinetics モジュールに移行できると考えられる。

- marble.solvercore.kinetics.PointKinetics
  - 標準解法
    - PointKineticsPade クラス（行列指数法（Padé 近似））
    - PointKineticsExplicitIntegration クラス（陽解法）
    - PointKineticsImplicitIntegration クラス（陰解法）
    - PointKineticsThetaIntegration クラス（θ 法）
    - PointKineticsRungeKuttaIntegration クラス（ルンゲクッタ法）
    - PointKineticsTimeIntegration クラス（時間積分法）
  - 即発跳躍近似
    - PointKineticsExplicitIntegrationPromptJump クラス（陽解法＋即発跳躍近似）
    - PointKineticsImplicitIntegrationPromptJump クラス（陰解法＋即発跳躍近似）
    - PointKineticsThetaIntegrationPromptJump クラス（θ 法＋即発跳躍近似）
    - PointKineticsRungeKuttaIntegrationPromptJump クラス（ルンゲクッタ法＋即発跳躍近似）
    - PointKineticsTimeIntegrationPromptJump クラス（時間積分法＋即発跳躍近似）

  - 補助関数
    - pade 関数（Padé 近似による行列指数の計算）
    - explicit_integration 関数（陽解法による積分の計算）
    - implicit_integration 関数（陰解法による積分の計算）
    - theta_integration 関数（θ 法による積分の計算）
    - rungekutta_integration 関数（ルンゲクッタ法による積分の計算）
- marble.solvercore.kinetics.PointKineticsAnaliticSolution
  - PointKineticsAnalyticSolutionForStepReactivity クラス（厳密解（ステップ入力））
  - PointKineticsLinearApproximationForRampReactivity クラス（線形近似解（ランプ入力））
  - PointKineticsLinearApproximationForSinusoidalReactivity クラス（線形近似解（正弦波入力））
- marble.solvercore.kinetics.InversePointKinetics
  - InversePointKinetics クラス（逆解法）
- marble.solvercore.kinetics.KineticsUtils
  - KineticsParameter クラス（動特性パラメータの設定）

3.3.3 検証

今回の再実装では、Pointkinetics モジュールの単体テストを基本的に全て再現することを確認し、Kinetics モジュールの単体テストとして改めて整備した。なお、Pointkinetics モジュールの単体テストでは、一点炉動特性ソルバーの標準解法となる行列指数法（Padé 近似）のソルバーが正常に動作することを、参照解のソルバーの結果と比較することにより確認している。その上で、行列指数法（Padé 近似）のソルバーの結果を参照解として、各種の解法に基づくソルバーが参照解と同等の結果を出すことを確認している。Kinetics モジュールの単体テストでも、同様の方法でテストを行っている。

また、再実装した Kinetics モジュールの参照解のソルバーに関しては、Pointkinetics モジュールの参照解のソルバーと比較を行い、Pointkinetics モジュールの参照解のソルバーと同等の結果が得られることを確認した。

3.3.4 補足：逆時間方程式について

Pointkinetics モジュール及び Kinetics モジュールにおいて参照解を計算するソルバーでは、以下の特性方程式（逆時間方程式）（本文中の式 (3.30)）を数値的に解く必要がある。

$$\Lambda\omega + \beta - \rho - \sum_i \frac{\beta_i \lambda_i}{\omega + \lambda_i} = 0 \tag{3.66}$$

Pointkinetics 報告書の (2-24) 式

この式を ρ について整理すると以下のようになる。

$$\rho = \Lambda\omega + \beta - \sum_i \frac{\beta_i \lambda_i}{\omega + \lambda_i} \tag{3.67}$$

一方で、文献 [51,52] に示されている逆時間方程式は以下のようになっている。

$$\rho = \omega\Lambda + \omega \sum_i \frac{\bar{\beta}_i}{\lambda_i + \omega} \tag{3.68}$$

文献 [51] の (2-24) 式

$$\rho_1 = \omega_j \Lambda + \sum_i \frac{\omega_j \beta_i}{\omega_j + \lambda_i} \tag{3.69}$$

文献 [52] の (58) 式

この二つの式は記号の定義が異なっているだけで本質的には等価である。ω_j は $\sum_i$ の外に出してもよいことを考慮して、本メモの記号で書き直すと以下のようになる。

$$\rho = \Lambda\omega + \omega \sum_i \frac{\beta_i}{\omega + \lambda_i} \tag{3.70}$$

式 (3.67) と式 (3.70) は少し異なった形をしているが、$\beta = \sum_i \beta_i$ であることを考慮すると、以下のように、両者は等価な式であることが確認できる。

$$\begin{aligned}
\text{式 (3.67) の右辺} &= \Lambda\omega + \beta - \sum_i \frac{\beta_i \lambda_i}{\omega + \lambda_i} \\
&= \Lambda\omega + \sum_i \beta_i - \sum_i \frac{\beta_i \lambda_i}{\omega + \lambda_i} \\
&= \Lambda\omega + \sum_i \frac{\beta_i(\omega + \lambda_i) - \beta_i \lambda_i}{\omega + \lambda_i} \\
&= \Lambda\omega + \sum_i \frac{\beta_i \omega}{\omega + \lambda_i} \\
&= \Lambda\omega + \omega \sum_i \frac{\beta_i}{\omega + \lambda_i} = \text{式 (3.70) の右辺}
\end{aligned} \tag{3.71}$$

なお、逆時間方程式の解については、文献 [52] に詳しい解説がある。遅発中性子先行核が N 群の場合、逆時間方程式の解は $(N+1)$ 個存在し、この解を $\omega_0, \omega_1, \cdots, \omega_N$ とする（ただし、$\omega_0 > \omega_1 > \cdots > \omega_N$）と、以下の関係式が成り立つ。

$$-\lambda_1 < \omega_0 \tag{3.72}$$

$$-\lambda_{j+1} < \omega_j < -\lambda_j \quad (j = 1, 2, \cdots, N-1) \tag{3.73}$$

$$\omega_N < -\lambda_N \tag{3.74}$$

Kinetics モジュールでは、この関係式を使って逆時間方程式の解の探索を行っている。

4. おわりに

汎用炉心解析システム MARBLE の第 3 版（MARBLE3）では、開発で利用している Python のバージョンアップに伴う後方非互換性の問題を解決するために、全面的に改修を行って、Python バージョン 3（Python3）で動作するように整備した。また、MARBLE の第 2 版（MARBLE2）以降に開発された 3 次元 Hex-Z/Tri-Z 体系の輸送計算コード MINISTRI Ver.7（MINISTRI7）や 3 次元 Hex-Z/三角 Tri-Z 体系の拡散計算コード D-MINISTRI の組み込みを行い、MARBLE のサブシステムとして組み込まれている高速炉核特性解析システム SCHEME や高速炉燃焼解析システム OPRHEUS で利用できるように整備した。MARBLE2 でも 3 次元 Hex-Z 体系の輸送計算コード NSHEX を利用することができたが、収束性の問題等から、計算用メッシュを 3 次元 Hex-Z 体系から 3 次元 XYZ 体系に変換して、3 次元 XYZ 体系輸送計算コード TRITAC で計算することが多かった。しかしながら、今後は MARBLE3 と MINISTRI7 を利用して、直接、3 次元 Hex-Z/Tri-Z 体系で輸送計算することが現実的になると考えられる。

また、MARBLE3 では、MARBLE2 以降に提案された燃焼計算に関する解法として、チェビシェフ有理関数近似法（CRAM-PFD、CRAM-IPF）、ミニマックス多項式近似（MMPA）法を燃焼計算ソルバーに組み込んだ。一方で、メンテナンス性や柔軟性の観点から、MARBLE に組み込まれていた CBG システムのインターフェースを拡張して、CBG の 2 次元 RZ 体系拡散計算ソルバーや輸送計算ソルバーを利用できるように整備した。これらの整備により、高速炉の核特性解析でよく用いられる 2 次元 RZ 体系、3 次元 XYZ 体系、Hex-Z 体系、Tri-Z 体系に対して国産のコードのみで主要な解析を実施できるようになった。ただし、これらの新しい機能は、MARBLE が開発される以前から利用されていた従来の高速炉核特性解析システムにはなかった機能であるので、今後、検証や妥当性確認を行っていく必要がある。また、メンテナンス性の観点から、MARBLE2 で導入された一点炉動特性ソルバーを全面的に書き直した。

MARBLE の開発を開始した当初は、Python の利用者はそれほど多くなかったが、Python は、近年研究開発が盛んになっている機械学習や深層学習、強化学習等の分野で事実上の標準プログラミング言語となっており、Python の利用者は増大している。このため、Python 自体の進化も速くなっており、今後も Python の仕様変更に伴う MARBLE の改修が必要になると考えられるので、継続的に MARBLE のメンテナンス性の向上のための改良を行っていく必要があると考えられる。また、MARBLE3 では燃焼計算手法のアルゴリズムの改良版の導入等を行ったが、炉心解析手法自体も発展しているので、今後も新しい計算手法等の導入を進めていく必要がある。

一方で、Python の利用者が増えたことで、ライブラリの開発が進み、機械学習や深層学習、強化学習等の分野に限らず、多種多様な高機能なライブラリを利用することができるようになっている。MARBLE は Python のライブラリとして利用することが可能なので、Python で提供されている他のライブラリと組み合せて使うのは容易である。MARBLE3 と Python で提供されている様々なライブラリを組み合わせることで、MARBLE3 の核特性解析の機能と異なる分野の成果を

融合させた研究開発を効率的に行うことができると考えられる。

謝辞

（株）NESI の神智之氏には、Python3 への対応をはじめとして、PSAGEP の統合、MINISTRI7 等の新しい解析コードの導入や新規ソルバーの導入に伴う整合性の調整、パッケージの管理、継続的インテグレーションを伴う自動テストの管理等、MARBLE の開発に関して長年に亘ってご協力をいただきました。ここに記して感謝の意を表します。

その他、ユーザの方々から多くの有益なフィードバックをいただきました。ここに記して感謝の意を表します。

参考文献

1) 横山賢治, 細貝広視, 宇都成昭, 笠原直人, 名倉文則, 大平正則, 加藤雅之, 石川眞：“工学系モデリング言語としての次世代解析システムの開発 (I) – 課題および要素技術の調査 –”, JNC TN9420 2002-004 (2002), 296p.

2) 横山賢治, 細貝広視, 宇都成昭, 笠原直人, 石川眞：“工学系モデリング言語としての次世代解析システムの開発 (II) – プロトタイプ作成による検討 –”, JNC TN9400 2003-021 (2003), 187p.

3) 横山賢治, 細貝広視, 千葉豪, 笠原直人, 石川眞：“工学系モデリング言語としての次世代解析システムの開発 (III) – プロトタイプ作成による検討 (その 2) –”, JNC TN9400 2004-022 (2004), 151p.

4) 千葉豪：“工学系モデリング言語としての次世代解析システムの開発 (IV) – 境界要素法に基づく中性子拡散ソルバーの開発 –”, JNC TN9400 2004-055 (2004), 48p.

5) 横山賢治：“工学系モデリング言語としての次世代解析システムの開発 (V) – 二階層システム移行のための既存コード再構築手法 の検討 –”, JAEA-Research 2006-055 (2006), 60p.

6) 横山賢治, 沼田一幸：“工学系モデリング言語としての次世代解析システムの開発 (VI) – 炉定数調整・核設計精度評価ソルバーの開発 –”, JAEA-Data/Code 2007-023 (2008), 39p.

7) 横山賢治, 巽雅洋, 平井康志, 兵頭秀昭, 沼田一幸, 岩井武彦, 神智之, 羽様平, 長家康展, 千葉豪, 久語輝彦, 石川眞：“次世代炉心解析システム MARBLE の開発”, JAEA-Data/Code 2010-030 (2011), 148p.

8) 横山賢治, 神智之, 平井康志, 羽様平：“汎用炉心解析システム MARBLE2 の開発”, JAEA-Data/Code 2015-009 (2015), 120p.

9) 中川正幸, 阿部純一, 佐藤若英：“高速炉の核特性解析コードシステム”, JAERI-M 83-066 (1983), 74p.

10) 石川眞, 斎藤正幸, 佐藤若英, 三田敏男, 池上哲雄：“核設計基本データベースの整備 (IV) –核特性解析コードシステムの整備–”, PNC TN9440 94-004 (1994), 293p.

11) 原昭浩, 竹田敏一, 菊池康之：“SAGEP: 一般化摂動論に基づく二次元感度解析コード”, JAERI-M 84-027 (1984), 31p.

12) K. D. Lathrop and F. W. Brinkley: “TWOTRAN-II: An interfaced, exportable version of the twotran code for two-dimensional transport,” LA-4848-MS (1973), 71p.

13) 佐々木誠, 市川真一：“「常陽」輸送コードシステムの作成 S_N 輸送コード・使用マニュアル”, PNC TN952 81-08 (1981), 109p.

14) K. Yokoyama, T. Hazama, K. Numata and T. Jin: “Development of comprehensive and versatile framework for reactor analysis, MARBLE,” Ann. Nucl. Energy, vol. 66, pp. 51–60 (2014).

15) 深谷裕司：“次世代炉心解析システム MARBLE 用一点炉動特性ソルバー Pointkinetics の開発”, JAEA-Data/Code 2011-014 (2011), 55p.

16) 杉野和輝：“3 次元六角体系用中性子輸送計算コードの整備 (2) – MINISTRI コードの改良及び機能拡張 –”, JAEA-Data/Code 2019-011 (2019), 110p.

17) 杉野和輝：“3 次元六角体系用中性子輸送計算コードの整備 – NSHEX、MINIHEX、MINISTRI コードの整備 –”, JAEA-Data/Code 2011-018 (2012), 125p.

18) G. Chiba and Y. Shimazu: “Sodium void reactivity worth calculations for fast critical assemblies without whole-lattice homogenization,” J. Nucl. Sci. Technol., vol. 44, no. 12, pp. 1526–1534 (2007).

19) 飯島進, 吉田弘幸, 桜木廣隆：“高速炉設計用計算プログラム・2（2 次元・3 次元拡散摂動理論計算コード: PERKY）”, JAERI-M 6993 (1977), 51p.

20) T. Suenaga: “PyTC (python bindings of Tokyo Cabinet) on GitHub,” https://github.com/gunyarakun/pytc, (accessed 2024-05-01).

21) T. Veldhuizen: “Blitz++ user’s guide, a C++ class library for scientific computing for version 0.9, 24 March 2006,” (2006).

22) S. Arabas et al.: “Blitz++ multi-dimensional array library for C++,” https://github.com/blitzpp/blitz, (accessed 2024-05-01).

23) C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant: “Array programming with NumPy,” Nature, vol. 585, no. 7825, pp. 357–362 (2020).

24) P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and SciPy 1.0 Contributors: “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” Nature Methods, vol. 17, pp. 261–272 (2020).

25) M. Bando, T. Yamamoto, Y. Saito and T. Takeda: “Three-dimensional transport calculation method for eigenvalue problems using diffusion synthetic acceleration,” J. Nucl. Sci. Technol., vol. 22, no. 10, pp. 841–850 (1985).

26) 山本敏久：“炉心核特性詳細解析コード TRITAC の改良”, PNC TN9410 95-069 (1995), 65p.

27) T. B. Fowler, D. R. Vondy and G. W. Cunningham: “Nuclear reactor core analysis code: CITATION,” ORNL-TM-2496, Rev.2 (1971), 257p.

28) K. L. Derstine: “DIF3D: A code to solve one-, two- and three-dimensional finite-difference diffusion theory problems,” ANL-82-64 (1984), 292p.

29) A. G. Croff: “ORIGEN2 – a revised and updated version of the Oak Ridge isotope generation and depletion code,” ORNL-5621 (1980), 57p.

30) A. Yamamoto, M. Tatsumi and N. Sugimura: “Numerical solution of stiff burnup equation with short half lived nuclides by the Krylov subspace method,” J. Nucl. Sci. Technol., vol. 44, no. 2, pp. 147–154 (2007).

31) M. Pusa and J. Leppänen: “Computing the matrix exponential in burnup calculations,” Nucl. Sci. Eng., vol. 164, no. 2, pp. 140–150 (2010).

32) M. Pusa: “Rational approximations to the matrix exponential in burnup calculations,” Nucl. Sci. Eng., vol. 169, no. 2, pp. 155–167 (2011).

33) M. Pusa: “Higher-order chebyshev rational approximation method and application to burnup equations,” Nucl. Sci. Eng., vol. 182, pp. 297–318 (2016).

34) Y. Kawamoto, G. Chiba, M. Tsuji and T. Narabayashi: “Numerical solution of matrix exponential in burn-up equation using mini-max polynomial approximation,” Ann. Nucl. Energy, vol. 80, pp. 219–224 (2015).

35) 巽雅洋, 兵頭秀昭：“燃焼感度解析コードのシステム化整備”, JNC TJ9400 2003-012 (2004), 146p.

36) 巽雅洋, 兵頭秀昭：“燃焼感度解析コードのシステム化整備 (II)”, JNC TJ9410 2004-002 (2005), 89p.

37) K. Yokoyama, M. Ishikawa, M. Tatsumi and H. Hyoudou: “Restructuring of burnup sensitivity analysis code system by using an object-oriented design approach,” Proc. Int. Conf. on Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications, September 12–15, 2005 (M&C2005), Avignon, France (2005).

38) T. Takeda and T. Umano: “Burnup sensitivity analysis in a fast breeder reactor - Part I: Sensitivity calculation method with generalized perturbation theory,” Nucl. Sci. Eng., vol. 91, pp. 1–10 (1985).

39) 花木洋, 澤田周作, 三田敏男：“燃焼核特性に対する感度解析コードの整備”, PNC TJ9124 93-009 (1993), 334p.

40) 花木洋, 三田敏男, 大橋正久：“燃焼核特性に対する感度解析コードの整備 (II)”, PNC TJ9124 94-007 (1994), 723p.

41) T. Hazama, G. Chiba, W. Sato and K. Numata: “SLAROM-UF: Ultra fine group cell calculation code for fast reactor – Version 20090113 –,” JAEA-Review 2009-003 (2009), 59p.

42) 石川眞, 佐藤若英, 杉野和輝, 横山賢治, 沼田一幸, 岩井武彦：“核設計基本データベースの整備 (VIII) – JUPITER 実験解析結果の集大成 –”, JNC TJ9410 97-099 (1997), 512p.

43) G. Chiba and Y. Shimazu: “Sodium void reactivity worth calculations for fast critical assemblies without whole-lattice homogenization,” J. Nucl. Sci. Technol., vol. 44, no. 12, pp. 1256–1534 (2007).

44) R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner and R. C. Ward: “PARTISN: A time-dependent, parallel neutral particle transport code system,” LA-UR-08-07258 (2008), 516p.

45) 横山賢治, 神智之：“ORIGEN2 用断面積ライブラリセットとチェビシェフ有理関数近似法に基づく燃焼計算コード CRAMO の開発”, JAEA-Data/Code 2021-001 (2021), 47p.

46) E. Gallopoulos and Y. Saad: “Efficient solution of parabolic equations by Krylov approximation methods,” SIAM, J. Sci. Stat. Comput., vol. 13, no. 5, pp. 1236–1264 (1992).

47) M. Pusa: “Correction to partial fraction decomposition coefficients for Chebyshev rational approximation on the negative real axis,” arXiv:1206.2880v1 [math.NA] (2012).

48) A. Tsilanizara, N. Gilardi, C. J. T.D. Huynh, S. Lahaye, J. Martinez and C. Diop: “Probabilistic approach for decay heat uncertainty estimation using uranie platform and mendel depletion code,” Ann. Nucl. Energy, vol. 90, pp. 62–70 (2016).

49) K. Yokoyama and S. Lahay: “Benchamaks of depletion and decay heat calculation between mendel and marble,” Proc. of Joint Int. Conf. on Supercomputing in Nuclear Applications + Monte Carlo 2020 (SNA+MC2020), No. 3294033, pp. 109–116 (2020).

50) 片倉純一, 片岡理治, 須山賢也, 神智之, 大木繁夫：“JENDL-3.3 に基づく ORIGEN2 用断面積ライブラリセット: ORLIBJ33”, JAERI-Data/Code 2004-015 (2004), 115p.

51) 小林啓祐：“原子炉物理”, コロナ社 (1996).

52) 遠藤知弘：“動特性方程式の基礎理論”, 第 46 回炉物理夏期セミナーテキスト, pp. 24–61 (2014).

This is a blank page.