JNC TJ9410 2004-002

燃焼感度解析コードのシステム化整備(II)

（核燃料サイクル開発機構　契約業務報告書）

２００５年２月

原子燃料工業株式会社

〒319－1184　茨城県那珂郡東海村村松４番地４９
核燃料サイクル開発機構
技術展開部　技術協力課
電話：029-282-1122（代表）
ファックス：029-282-7980
電子メール：jserv@jnc.go.jp

JNC TJ9410 2004-002
2005 年 2 月

燃焼感度解析コードのシステム化整備(II)
（核燃料サイクル開発機構　契約業務報告書）

巽　雅洋※，兵頭　秀昭※

要　旨

高速炉の実用化に向けて、高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも、信頼性および安全性の裕度をより高める上でも、極めて重要な研究課題となっている。

これまでの研究では、炉定数調整法を適用することにより、JUPITER 等の臨界実験の成果を最大限有効に反映した統合炉定数を開発し、核設計精度の大幅な向上を達成している。一方、高速炉の炉心設計に於いては、臨界性、反応率、制御棒価値等のいわゆる静核特性だけでなく、燃焼反応度損失、増殖比といった燃焼核特性の精度良い評価も重要である。このためには、高速実験炉「常陽」等の豊富な実機燃焼データを有効に活用して、燃焼核特性の精度向上を図る必要がある。

炉定数調整法により、実機燃焼データを活用するためには、燃焼核特性の感度解析（以下、燃焼感度解析と呼ぶ）を行う必要がある。これまでに、燃焼感度解析を実施するためのコード（SAGEP-BURN）の開発が行われ、その有効性が確認されている。しかしながら、この燃焼感度の理論の複雑さと、システム上の制限から、ユーザへの負担が大きく解析作業が極めて非効率的であるという問題があった。また、システムの巨大化により機能の拡張が難しくなっているため、今後の機能拡張のために整理・統合が必要となっている。

一方、解析対象によって計算ステップが変わることや、物理的意味を分析する際には計算ステップを分解する必要があること等から、各計算機能を単純に統合するだけでは不十分である。各計算ステップは部品として保持したまま、必要に応じて部品を組み立てたり分解したりできるようにして、現在の燃焼感度解析コードをシステム化する必要がある。このため、オブジェクト指向とスクリプト言語の技術を利用して、燃焼感度解析コードのシステム化作業を実施した。

本研究では、オブジェクト指向スクリプト言語 Python を用いた既存システムの二階層制御モデルについて検討を行った。その成果を元に、新燃焼感度解析システム PSAGEP (Python-wrapped SAGEP-burn)を実装した。これより、二階層制御モデルによる既存システムの再構築手法が、有効であることが確認された。

※原子燃料工業（株）

JNC TJ9400 2004-002

JNC TJ9410 2004-002

February 2004

Systemization of Burnup Sensitivity Analysis Code (II)
(Document Prepared by Other Organization, Based on the Contract)

Masahiro Tatsumi※ and Hideaki Hyoudou※

Abstract

Towards the practical use of fast reactors, it is a very important subject to improve prediction accuracy for neutronic properties in LMFBR cores from the viewpoint of improvements on plant efficiency with rationally high performance cores and that on reliability and safety margins.

A distinct improvement on accuracy in nuclear core design has been accomplished by the development of adjusted nuclear library using the cross-section adjustment method, in which the results of criticality experiments of JUPITER and so on are reflected. In the design of large LMFBR cores, however, it is important to accurately estimate not only neutronic characteristics, for example, reaction rate distribution and control rod worth but also burnup characteristics, for example, burnup reactivity loss, breeding ratio and so on. For this purpose, it is desired to improve prediction accuracy of burnup characteristics using the data widely obtained in actual core such as the experimental fast reactor "JOYO".

The analysis of burnup characteristics is needed to effectively use burnup characteristics data in the actual cores based on the cross-section adjustment method. So far, a burnup sensitivity analysis code, SAGEP-BURN, has been developed and confirmed its effectiveness. However, there is a problem that analysis sequence become inefficient because of a big burden to users due to complexity of the theory of burnup sensitivity and limitation of the system. It is also desired to rearrange the system for future revision since it is becoming difficult to implement new functions in the existing large system

It is not sufficient to unify each computational component for the following reasons; the computational sequence may be changed for each item being analyzed or for purpose such as interpretation of physical meaning. Therefore, it is needed to systemize the current code for burnup sensitivity analysis with component blocks of functionality that can be divided or constructed on occasion. For this purpose, the burnup sensitivity analysis code has synthesized with an object-oriented scripting language.

In the present study, an examination was conducted for the two-layer controlling model of the conventional system using Python, the object-oriented scripting language. On the basis of the result in the examination, a new analysis system for burnup sensitivity, PSAGEP (Python-wrapped SAGEP-burn), was implemented. It was confirmed the effectiveness of the reconstruction method based on the two-layer controlling model of

---

※ Nuclear Fuel Industries, Ltd.

目　次

表リスト

図リスト

1. はじめに

高速炉の実用化に向けて、高速炉実機炉心の核特性予測精度を向上させることは、合理的で高性能な炉心を設計してプラントの経済性の向上を図る上でも、信頼性および安全性の裕度をより高める上でも、極めて重要な研究課題となっている。

これまでの研究では、炉定数調整法を適用することにより、JUPITER 等の臨界実験の成果を最大限有効に反映した統合炉定数を開発し、核設計精度の大幅な向上を達成している。しかし、高速炉の炉心設計に於いて、臨界性、反応率、制御棒価値等のいわゆる静核特性だけでなく、燃焼反応度損失、増殖比といった燃焼核特性の精度良い評価も重要である。このためには、高速実験炉「常陽」等の豊富な実機燃焼データを有効に活用して、燃焼核特性の精度向上を図る必要がある。

炉定数調整法により、実機燃焼データを活用するためには、燃焼核特性の感度解析（以下、燃焼感度解析と呼ぶ）を行う必要がある。これまでに、燃焼感度解析を実施するためのコード[1,2]（SAGEP-BURN）の開発が行われ、その有効性が確認されている。しかしながら、この燃焼感度の理論では、燃焼に伴う時間ステップを逆の方向に解いたり、物理的意味を持つ複数の項毎に計算したりする必要があるため、現在の燃焼感度解析コードシステムは、理論に合わせてユーザが計算ステップをひとつずつ追いかけながら使うようになっており、ユーザへの負担が大きく解析作業が極めて非効率的である。また、システムの巨大化により機能の拡張が難しくなっているため、今後の機能拡張のために整理・統合が必要となっていた。

一方、解析対象によって計算ステップが変わることや、物理的意味を分析する際には計算ステップを分解する必要があること等から、各計算機能を単純に統合するだけでは不十分である。各計算ステップは部品として保持したまま、必要に応じて部品を組み立てたり分解したりできるようにして、現在の燃焼感度解析コードをシステム化する必要がある。

なお、サイクル機構が進める次世代解析システムの開発[5]では、既存の解析システムをどのようにして、次世代解析システムの持つべき基本構成（制御層と計算層の二階層からなるシステム構成と汎用のオブジェクト指向スクリプト言語の採用）に移行するかが課題のひとつとなっている。本システム化整備は、燃焼感度解析コードをシステム化して整備することが第一の目的であるが、次世代解析システムの開発の観点からは、既存の旧システム構成を新システム構成に移行するための方法を検討するという第二の目的を持っている。

昨年度においては、オブジェクト指向とスクリプト言語の技術を利用して、燃焼感度解析コードをシステム化する作業を実施した[3]。オブジェクト指向スクリプト言語としてはPythonを採用し、Fortranで記述された既存システムの制御層の設計と実装を行った。これにより、既存のSAGEP-BURNシステムの全機能はPython言語[4]レベルで制御可能となり、入出力に関わる部分の柔軟性が大幅に向上した。

本年度の開発においては、ユーザによる解析を支援する観点から、必要な入力データフォーマットや解析シナリオの実現方法について検討を行った。この検討結果を元に、新システムである

PSAGEP(Python-wrapped SAGEP-burn)を開発した。同システムは、既存システムに比べ、より柔軟に解析が実施できること確認した。

2. PSAGEP システムの詳細設計

本章では、まず燃焼感度解析コード SAGEP-BURN のシステム化整備の概要について述べる。その後、同システムの設計に関して、昨年度における検討結果とその改善策についての検討結果を示す。また、SAGEP-BURN に関する入力情報整理し、計算コード間や解析ケース間での入力情報の共有について検討するとともに、システムの動作原理について検討を行った。

2.1 SAGEP-BURN のシステム化整備の概要

SAGEP-BURNシステムは、表 2-1 に示す燃焼感度解析のためのシステムモジュール群（PSAGEPシステムにおけるバックエンド計算コード）から構成された巨大なシステムとなっている。各システムモジュールは独立しており、それぞれに入力ファイルを指定する必要がある。また、モジュール間の情報のやり取りは、ファイルI/Oを介して行われ、全体の制御はシェルスクリプトで行われていた。一例として、図 2-1 に増殖比に関する計算フローの一部を示すが、解析フローは汎用性と引き換えに複雑なものとなっており、計算のためのスクリプト生成が困難であった。また、解析対象となる燃焼特性ごとに計算フローが異なるため、別の燃焼特性に着目するためにはシェルスクリプトの大幅な変更も必要なため、煩雑であった。そこで、システム化にあたって、SAGEP-BURNシステムの問題点を抽出し、入出力ファイルとシステムモジュールの制御方法に関する改善案を検討した。その結果、オブジェクト指向スクリプト言語Pythonを用いた二層制御方式によるシステム化整備が最も有望であるとの結論を得た。二階層御方式とは、システムモジュールをPythonからの取り扱いを可能とするカプセル化層と、カプセル化されたモジュールオブジェクトを取り扱う制御層の二階層に分類する方式であり、システム化のための方法の一種である[5]。

システム開発は、2003 年度のフェーズ 1 と、2004 年度のフェーズ 2 にわたって実施した。フェーズ 1 においては、Pythonによるカプセル化層の設計・実装を行い、22 のシステムクラスと、それらに対応するテストクラスを実装した。これらの開発には、効率性を高めるために、いわゆる「アジャイル*開発」を採用している。これは、反復的でインクリメンタルな開発サイクル[6,7] であり、近年注目されている。同手法では、開発速度を速めると同時にソフトウェアの信頼性や保守性を高めるために、「テスト・ファースト・プログラミング」(Test First Programming)と呼ばれるテスト戦略[8]を採用した。これは、作成対象コードの動作イメージであるテストプログラムを先に作成し、その後に本体側を実装するという方法である。テストでは、各クラスの動作確認をおこなうユニット・テストと、複数オブジェクトの相互関係をテストする結合テストを実装した。フェーズ 1 開発において実装したカプセル化層の動作確認

* Agile: 「俊敏な」「機敏な」の意

は、増殖比の燃焼感度解析を例に、予め用意した参照ケースと同一の計算結果を再現することで行った。

フェーズ２開発においても、アジャイル開発手法を採用し、制御層に係わる部分の実装を行った。制御層を実装するに当たり、解析シーケンスの制御方法や、入力メタデータの管理方法について検討を行った。これに際し、ユーザは既存システム部をそれほど意識せずとも、PSAGEP システムによる解析が実施できるように留意した。以下では、これらのフェーズ 2 開発における検討事項について述べる。

表 2-1　SAGEP-BURN におけるシステムモジュールの一覧

ｼｽﾃﾑﾓｼﾞｭｰﾙ名	機　　能
ＳＴＡＲＴ－ＵＰ	マクロ断面積を作成する。
ＣＩＴＡＴＩＯＮ	拡散計算コード 中性子束、および随伴中性子束を求める。
ＦＩＲＥ－１	燃焼計算コード 中性子束の出力規格化因子を計算する。また、燃焼方程式を解き、原子数密度の時間変化を求めること、随伴燃焼方程式を時間的に逆に解き、随伴原子数密度を求めることが出来る。
ＦＩＲＥ－２	燃焼計算コード ＦＩＲＥ－１で計算される出力規格化因子を入力して、燃焼方程式を解き着目集合体の原子数密度を求める。
ＦＩＲＥ－３	燃焼計算コード 原子数密度の感度係数を求める時に、ＦＩＲＥ－１で計算される出力規格化因子を入力して、随伴燃焼方程式から随伴原子数密度を求める。
ＮＳ－ＩＮＩ	原子数密度の感度係数を求める時に随伴原子数密度の初期値を求める。
ＮＳ－ＩＮＩ２	実効増倍率の感度係数を求める時に随伴原子数密度の初期値を求める。
ＮＳ－ＩＮＩ３	燃焼反応度損失の感度係数を求める時に随伴原子数密度の初期値を求める。また、燃料交換による不連続性を考慮する時にも使用する。
ＮＳ－ＩＮＩ４	増殖比の感度係数を求める時に随伴原子数密度の初期値を求める。
ＮＳ－ＩＮＩ５	制御棒価値の感度係数を求める時に随伴原子数密度の初期値を求める。
ＮＳ－ＩＮＩ６	反応率の感度係数を求める時に随伴原子数密度の初期値を求める。
ＮＳ－ＪＵＭＰ	時間ステップの前後での随伴原子数密度の不連続性の計算を行い、次のステップの随伴原子数密度の値を求める。
ＲＥＦＵＥＬ	燃料交換を含めた原子数密度の管理を行う。
ＲＥＦＵＥＬ２	燃料交換を含めた随伴原子数密度の管理を行う。
ＲＥＦＵＥＬ３	燃焼反応度損失の場合に随伴原子数密度の不連続性を考慮するとともに、燃料交換を含めた随伴原子数密度の管理を行う。
ＳＣＧＩＶＥ	随伴出力Ｐ*を計算し感度係数の第２、５項（原子数密度項、出力項）を計算する。また、随伴一般化中性子束を計算するためのソース項の計算も行う。
ＳＡＧＥＰ９３	感度係数の第１、３項（直接項、中性子束項）を計算する。また増殖比と反応率の感度係数を求める場合、随伴出力、ソース項の計算を行う。

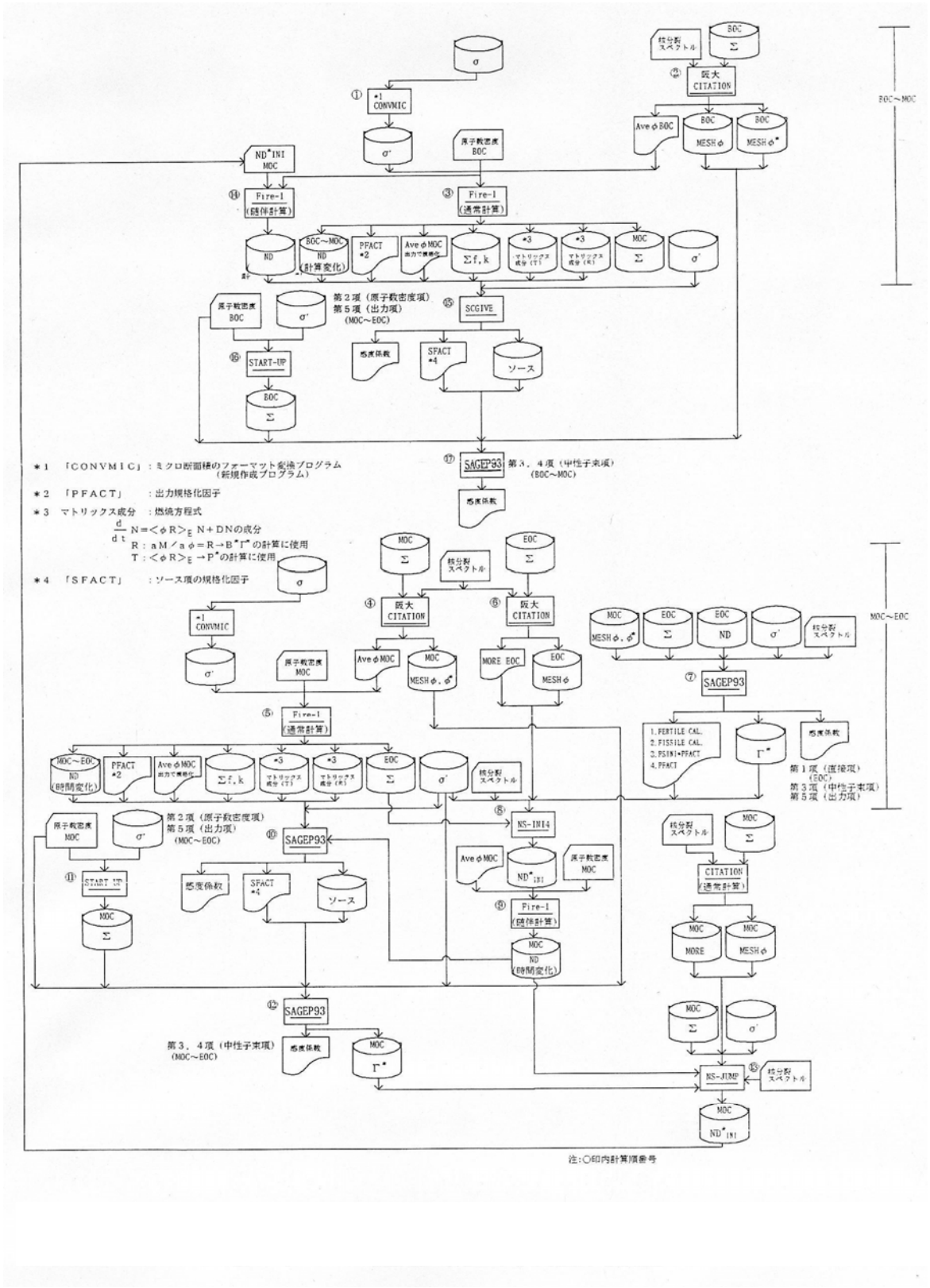


図 2-1　増殖比に関する感度係数の計算フロー（文献 2 から抜粋）

2.2 フェーズ1における設計案の再検討

カプセル化層の設計・実装であるフェーズ１が終了した時点において、フェーズ２における制御層のあり方について検討し、図 2-2 に示すPSAGEPシステムのクラス図[9]を作成した。本図は、次の基本方針に基づき導かれたものである。

- 入力中における情報は、適切な単位にまとめられ、クラス化されること
- それらはInfoクラスに汎化[9]されること
- Info クラスのインスタンスをまとめる InfoSet クラスを定義すること
- InfoManager は InfoSet クラスのインスタンスを保持すること

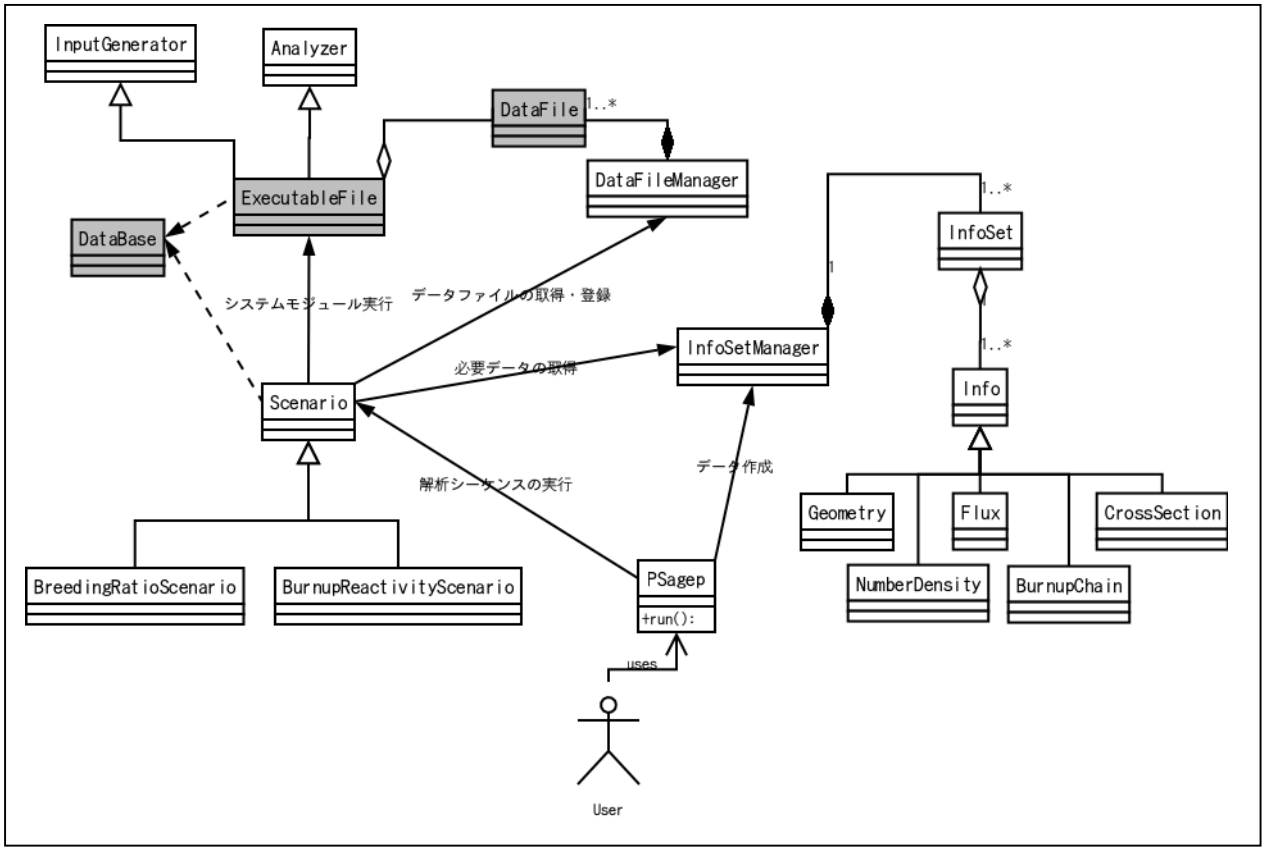


図 2-2　平成 15 年度報告書におけるフェーズ 2 設計案（クラス図）

フェーズ２の開発に際して、本設計の更なる検討を行った結果、次の疑問点が発生した。

- Info クラスのインスタンスを InfoSet としてとりまとめる必要があるか？
- InputGenerator は ExecutableFile の汎化として実現できるか？　それよりむしろ、DataFileManager 等のように、「管理クラス」として独立させた方が良いのではないか？
- 計算結果を加工する Analyzer クラスについても、InputGenerator クラスと同様の立場でよいか？

そこで、クラス間の関係について再検討を行い、次の方針にてクラス図を再検討することとした。

- InfoSet を省略し、全ての情報は直接的に InfoManager が管理する。
- InputGenerator と Analyzer は独立した「制御クラス」として分離する。

その結果、オブジェクトクラスを、メタ情報関連クラス(Info 系)、ファイル関連クラス(File 系)、管理クラス(Manager 系)の 3 つに分類した。表 2-2 に、各分類における代表クラスをまとめる。

表 2-2　オブジェクトクラスのカテゴリと代表クラス

クラス・カテゴリ	クラス名	役割
メタ情報関連クラス	Info	抽象データクラス
	Nuclide	核種の表現
	Material	核種と数密度
	Region	領域情報
	Geometry	体系情報
	BurnupChain	燃焼チェーン
	SensitivityOption	感度の解析条件
	ConvergenceCondition	収束条件
ファイルクラス	DataFile	一般データファイル
	ExecutableFile	実行モジュールカプセル化
	DataBase	データベース
管理クラス	Scenario	解析シナリオ
	InfoManager	入力情報管理
	InputGenerator	システムモジュールへの入力の作成
	DataFileManager	データファイル管理
	OutputManger	結果ファイル生成

また、標準的な解析シーケンスにけるクラス間の関わりについての検討を通じて、図２に示すクラス図を導出した。これをもとに、制御層の各クラスの詳細設計および実装を行った。

メタ情報関連クラスは、各種計算条件を保持するクラスであり、ユーザによる入力ファイルの中で具体的な内容が定義される。ファイル関連クラスは、データファイルや実行ファイルをオブジェクトでカプセル化するためのものである。また、管理クラスは、各種情報やファイルに関するオブジェクトを取り扱い、解析プロセスの制御を行うものである。二階層モデリングに照らし合わせると、メタ情報関連クラスとファイル関連クラスが「カプセル化層」に対応し、管理クラスが「制御層」に対応する。

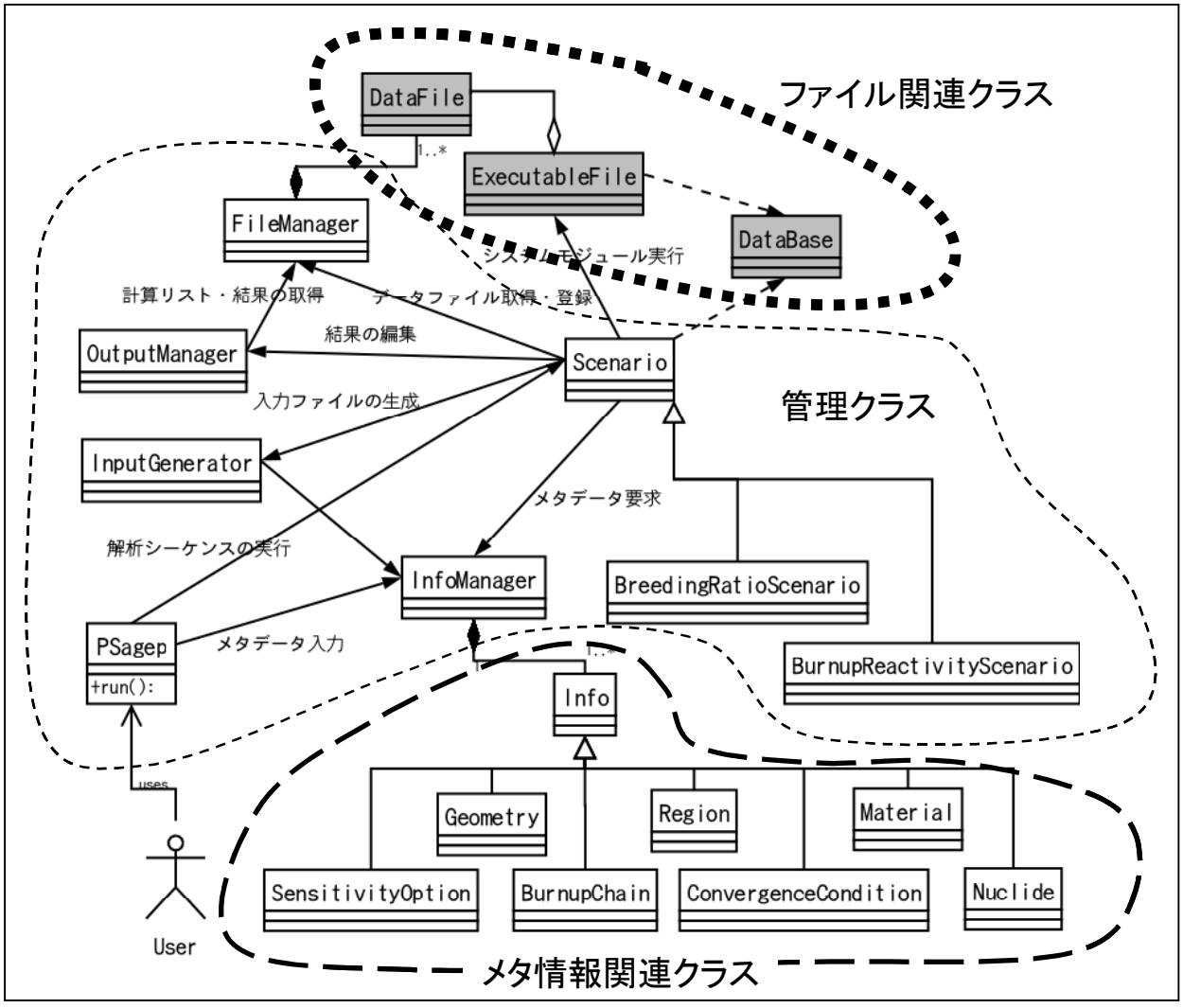


図 2-3　再検討後におけるフェーズ２システムのクラス図の設計

2.3 システムの動作原理の検討

フェーズ２システムにおいては、既存の動作原理を完全に隠蔽することが重要となる。すなわち、既存システムに関して、入力ファイルや出力ファイルの詳細仕様について理解しなくても、燃焼感度解析の方法論が理解できていれば、容易に解析ができるような仕組みを提供するべきである。したがって、既存システムとは切り離して、解析に必要なデータの入力方法を新規に検討する必要がある。

本節では、このような背景のもと、各燃料感度解析ケースにおいて、どのような入力情報が必要であるかを既存の解析事例をもとに検討した。そして、計算コード間、あるいは解析ケース間において、各システムモジュールに対する入力情報の共通性について検討した。

2.3.1 入力メタ情報

計算を制御する観点から、ユーザから与えられた解析条件が、各サブシステム（計算コード）が認識できるように自動的に加工される仕組みが必要である。そこでまず、増殖比に関する解析シナリオを対象として、各計算コードが必要とする情報について調査した。結果を表 2-3 にまとめる。なお、表には convmic 等の計算コード名が記載されているが、各コードの概要については文献 3 を参照されたい。

以下では、ユーザが指定すべき入力情報を、「入力メタ情報」と呼ぶこととする。表 2-3 によると、解析に必要な入力メタ情報として、次のものが挙げられる。

- 核種の指定（使われる核種の種類と数、PDS ファイル名）
- 領域数（サブ領域、軸方向分割数、交換集合体数）
- マテリアル（原子数密度、マテリアル領域数）
- 計算条件（メッシュ分点、メッス分割数、境界条件、収束条件、エネルギー群数）
- 燃焼条件（サイクル数、燃焼核種数、冷却日数、崩壊定数、燃焼チェーンマトリクス、炉出力、燃焼時間、燃焼サブステップ、着目ステップ数）
- その他（コード固有オプション、解析ケース固有オプション）

従来システムでは、これらの情報が各システムモジュール、解析ケース、燃焼サイクル毎に分散され、個別の入力ファイルとして作成する必要があった。したがって、あるパラメータを変更したい場合には、その情報が含まれる全てのファイルを修正する必要があり、管理が困難であるという問題があった。そのため、新システムでは、入力メタ情報は一箇所で集中管理し、各システムモジュールには必要に応じて情報が自動的に伝達される仕組みが必要となる。しかも、その情報伝達の仕組みについては、ユーザが意識する必要がないようにすることとした。

表 2-3　計算コードへの入力メタ情報（増殖比解析シナリオの場合）

↓項目　　計算コード→	convmic	pdsedit	refuel	startup	citation	fire1	fire2	nsini4	nsjump	refuel2	scgive	sagep93
核種数	○ 33		○ 33	○ 33								
核種ID（JENDL）	○ 925など		○ 925					○ 925				
領域数		○ 16	○ 16	○ 16		○ 16	←				○ 16	○ 16
サブ領域数			○ 4									
軸方向領域数			○ 4									
領域の(交換)集合体数			○ 1							○ 1.0など		
エネルギー群数		○ 18		○ 18		○ 18	←				○ 18	○ 18
PDSファイル名		○ IC01 等										
サイクル数設定			○ 1など							○ 1など		
原子数密度			○ 集合体毎									
燃焼核種数			○ 21			○ 21	←	○ 21	○ 21		○ 21	
崩壊定数			○ 2E-09							○ 2E-09		
冷却日数			○ 42							○ 42		
計算コード固有オプション												
反復回数					○ *1			○ *1	○ *1			○ *2 「
収束条件					○ *1			○ *1	○ *1			○ *2 「
マテリアル領域数					○ *1			○ *1	○ *1			
メッシュ分割数					○ *1			○ *1	○ *1			○ *2
メッシュ幅					○ *1			○ *1	○ *1			○ *2
マテリアルマップ					○ *1			○ *1	○ *1			○ *1
境界条件					○ *1			○ *1	○ *1			
燃焼マトリックス情報						○	←	△ 一部のみ				○
熱出力						○ 4E+06	←		○ 4E+06			○
燃焼時間						○ 5E+07	←				○ 5E+07	
燃焼ステップ数							←					
燃焼サブステップ数												
着目ステップ数												
感度計算対象オプション												○

○：情報の供給が必要（備考として例を表示）　　*1 共有化可能　　*2 形式が異なる

2.3.2　入力メタ情報の管理と入力ファイル作成

前章で述べた入力メタ情報は、InfoManager インスタンスにより集中的に管理することとした。図 2-3 に示すように、各計算コードに対するの入力ファイルは、入力メタ情報を用いて InputGenerator インスタンスにより作成される。同インスタンスは、ターゲットとなる計算コードへの入力を作成する際、必要な情報を適宜 InputManager から取得し、計算コードが期待するフォーマットに変換して出力する。

同じ計算コードであっても、解析対象となる物理量が異なる別の解析シナリオでは、入力ファイルの内容が異なる場合がある。この問題は、解析シナリオの管理を司る Scenario クラスが InputGenerator インスタンスを制御することで解決する。例えば、InputGenerator クラスのメソッドを起動する際に、通常計算モードや随伴計算モードといった動作モードを指定することにより、より詳細な管理を行うこととした。

2.3.3　解析シナリオの管理

燃焼感度の解析では、着目する核特性が異なると、解析シーケンス自体も変化する。すなわち、使用する計算コードの種類や実行する順番が解析ケース間で異なる。また、計算コード間で接続される入出力ファイルの種類や内容が異なってくるため、全ての解析シーケンスを包含することは出来ない。従って、解析シーケンス毎に「シナリオ」を定義し、ここに管理することとする。すなわち、解析シナリオは Scenario クラスにより管理し、実際の解析シーケンスは Scenario クラスのサブクラス内にて定義する。

Scenarioクラス内[1]では、次の手順により解析シーケンスを実行する。

- InputGenerator の入力作成メソッドを起動し、入力ファイルを作成する。
- 計算コードに接続するファイルを FileManager から取得する
- ラッパークラスの run()メソッドを起動し、当該の計算コードを実行する。
- 計算によって得られた出力ファイルを FileManager に登録する

この手続きは、従来の解析で作成した実行シェルスクリプトに相当するが、計算コード間の入出力の依存関係は自動的に解決されるため、管理に必要な労力は大幅に削減される。

2.3.4　解析ケース間におけるの入力の類似性についての検討

既存システムでは、先述のように、各計算コードに対してサイクル数や燃焼初期・末期に応じた入力ファイルを与えている。現状では、それぞれが別々のファイルとして準備されているが、その内容の大部分は共通であり、入力メタデータから InputGenerator により動的に作成することが出来る。

[1] 実際には、Scenarioクラスのサブクラス

ここでは、増殖比に関する燃焼感度解析(BR ケース)を例にとり、計算コードの解析ケース間での入力ファイルの類似性について検討する。これにより、入力ファイル形式を分類し、InputGenerator クラスの実装に供することとする。

図 2-4～図 2-12 に、各計算コードに解析ケース間での類似性検討の結果を示す。解析ケース関して、B は燃焼初期(BOC)を、E は燃焼末期(EOC)を、adj は随伴計算を、数字はサイクル数をそれぞれ示す。図において、「×」となっている場所は、行と列が示すケース間で入力ファイルに類似性が見られなかったことを示す。また、「△」「○」「◎」については類似性を確認した。なお、全てのケースについて検討する代わりに、法則性が発見できた段階でそれ以後の検討を省略している。以下に、検討結果をまとめる。

表 2-4　各計算コードにおける解析ケース間の入力類似性の検討結果

計算コード	サイクル間での入力類似性の検討結果
Refuel	・第 1 サイクルと、第二サイクルとではそもそも入力形式が異なる ・燃料交換（燃交）と燃焼サイクルでも入力形式が異なる。 ・BOC ケース→燃料交換：IOPT=0　　EOC ケース→燃焼：IOPT=1
Citation	・エディットすべき領域ラベル名が異なるのみ ・Forward と Adjoint の差異も、随伴計算に関するオプションのみ異なる
Fire1	・エディットすべき領域ラベル名が異なるのみ ・Forward と Adjoint は特になし
Fire2	・エディットすべき領域ラベル名が異なるのみ
Startup	・無修正で使用可能
Nsini4	・入力ファイルは一つのみ
Scgive	・エディットすべき領域ラベル名が異なるのみ
Sagep	・エディットすべき領域ラベル名が異なるのみ
Refuel2	・最終サイクルからのサイクル数情報が異なるのみ ・BOC 及び EOC でのサイクル間では類似性がある。 ・ただし、Cyc04 EOC のみ大幅に異なる（ブロック追加）

2.3.5　解析シナリオ間での入力類似性

次に、各計算コードの入力に関して、解析シナリオ間での類似性について検討を行った。図 2-13～図 2-23 に、各計算コードの解析シナリオ間での類似性検討の結果を示す。表中のシナリオ名の意味は下記の通りである。なお、BOC ケースはここでは省略している。また、「×」等の記号の意味は、先の類似性検討と同様である。

表 2-5　解析シナリオ名と対象の物理量

シナリオ名	解析対象の物理量
br	増殖比
burnup	燃焼反応度損失
keff_eoc	実効増倍率（EOC）
nav_eoc	ナトリウムボイド反応度(EOC)
nd-focused	注目核種数密度
rr_eoc	反応率(EOC)

表 2-6　バックエンド計算コードにおける解析シナリオ間の入力類似性の検討結果

計算コード	解析シナリオ間での入力類似性の検討結果
Refuel	・ほぼ完全に共有化できる。 ・cyc01-BOC のみ、nd-focused が異なる。
Startup	・解析シナリオに依らず、完全に共有化できる。
Citation	・keff, nav, nd, rr については、バックリング計算を行うオプションが立っている ・しかしながら、すべてを共有化しても本質的には計算結果には影響がないと思われる
Fire1	・すべて同一
Fire1(adj)	・すべて同一 ・ただし、nd-focused では adjoint 計算は行わない
Fire2	・すべて同一
Nsini4	・入力ファイルは一つのみ
Refuel2	・すべて同一
Scgive	・ほぼ同一 ・ただし、nd-focused において、いくつかのパラメータの指定が異なる
Sagep-flx	・nd-focused において、収束条件のみが異なる ・ただし、これは統合可能
Sagep-direct	・領域ラベルが異なる場合があるが、基本的には同一

以上の検討結果より、各計算コードにおいて入力ファイルの解析シナリオ間での類似性が高いことを確認した。したがって、さらなる検討においては代表シナリオついてまず検討し、その他のシナリオに対応することとした。そこで、以下では、増殖比(BR)に関する解析シナリオを代表例として取り上げて検討する。

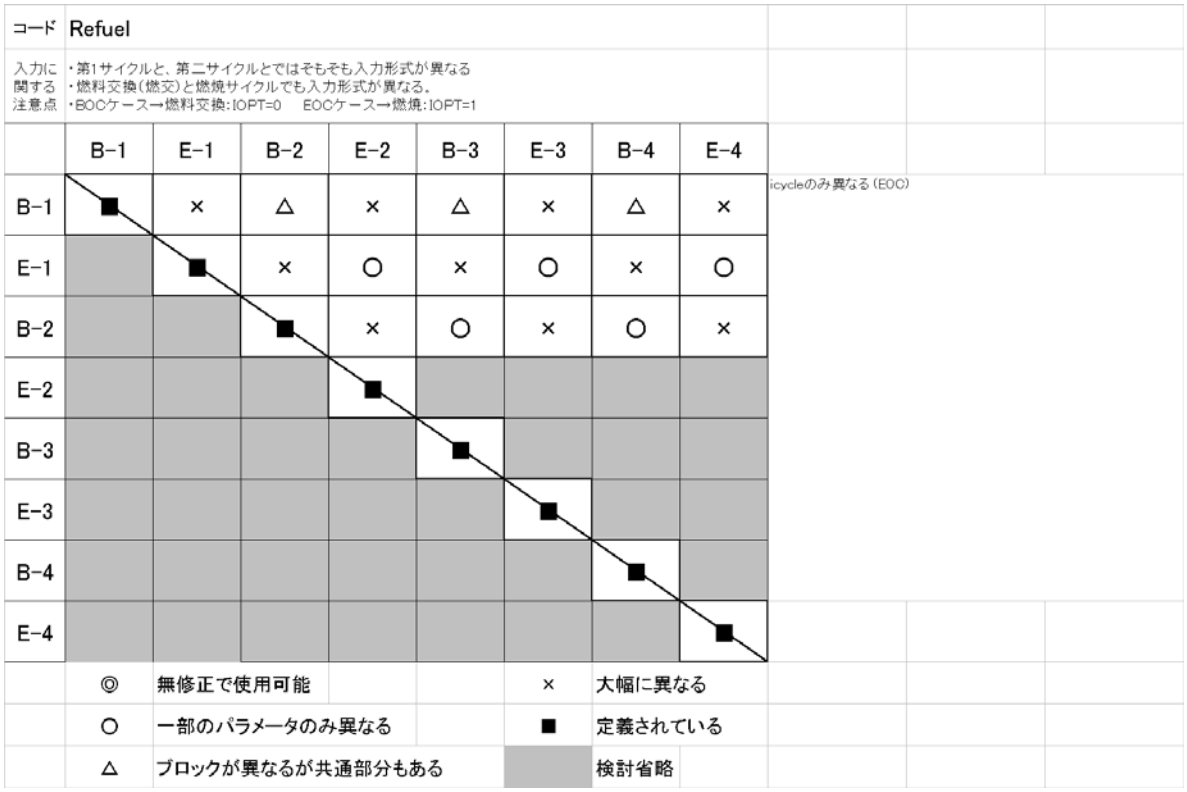


図 2-4　Refuel コードへの入力ファイルにおける解析ケース間での類似性検討

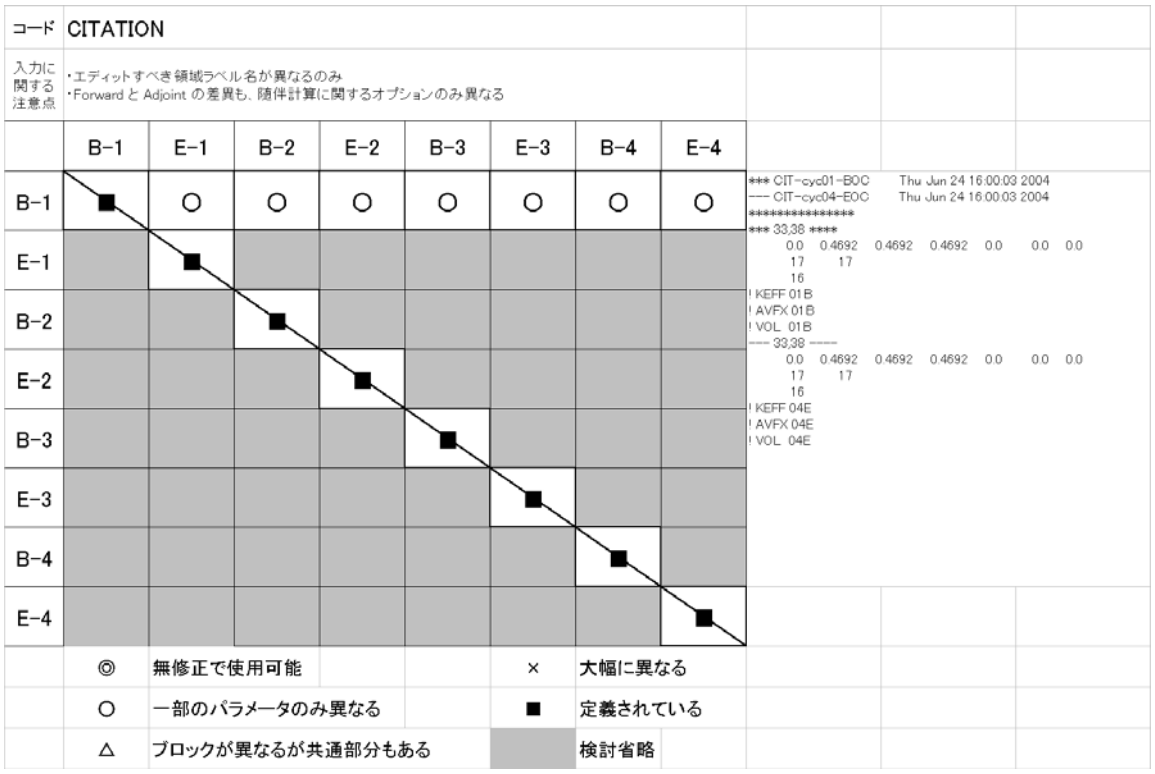


図 2-5　Citation コードへの入力ファイルにおける解析ケース間での類似性検討

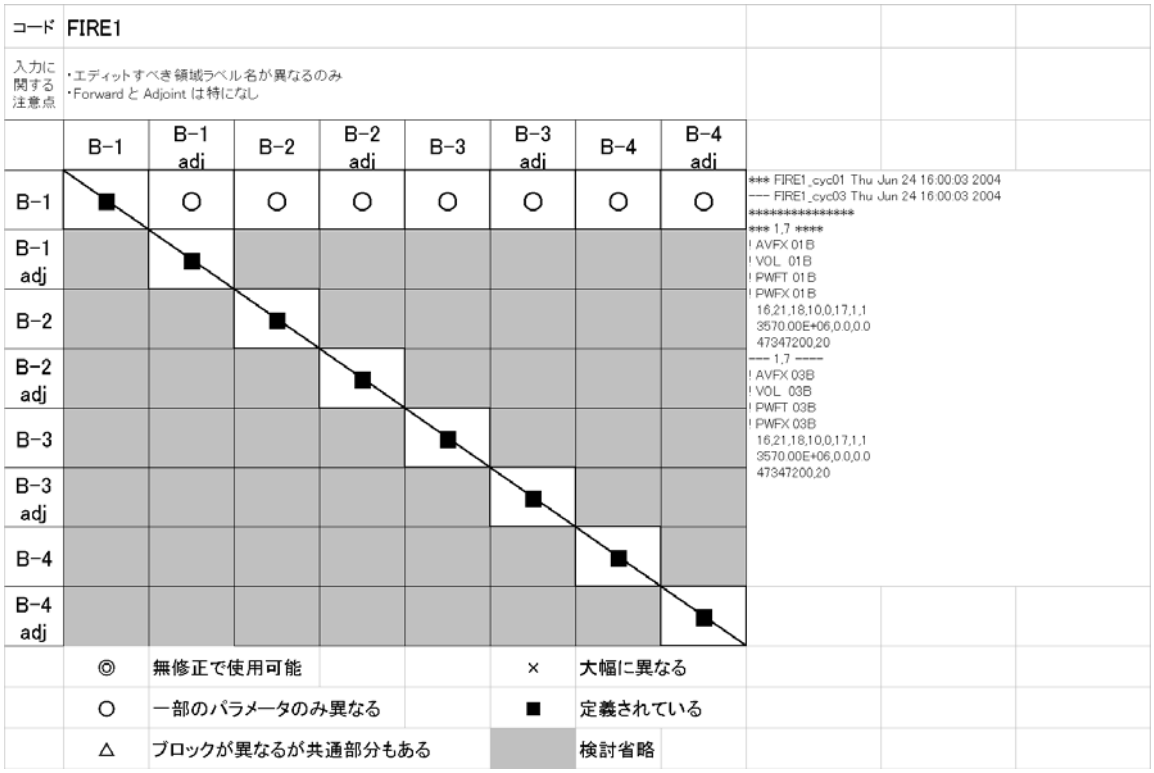


図 2-6　Fire1 コードへの入力ファイルにおける解析ケース間での類似性検討

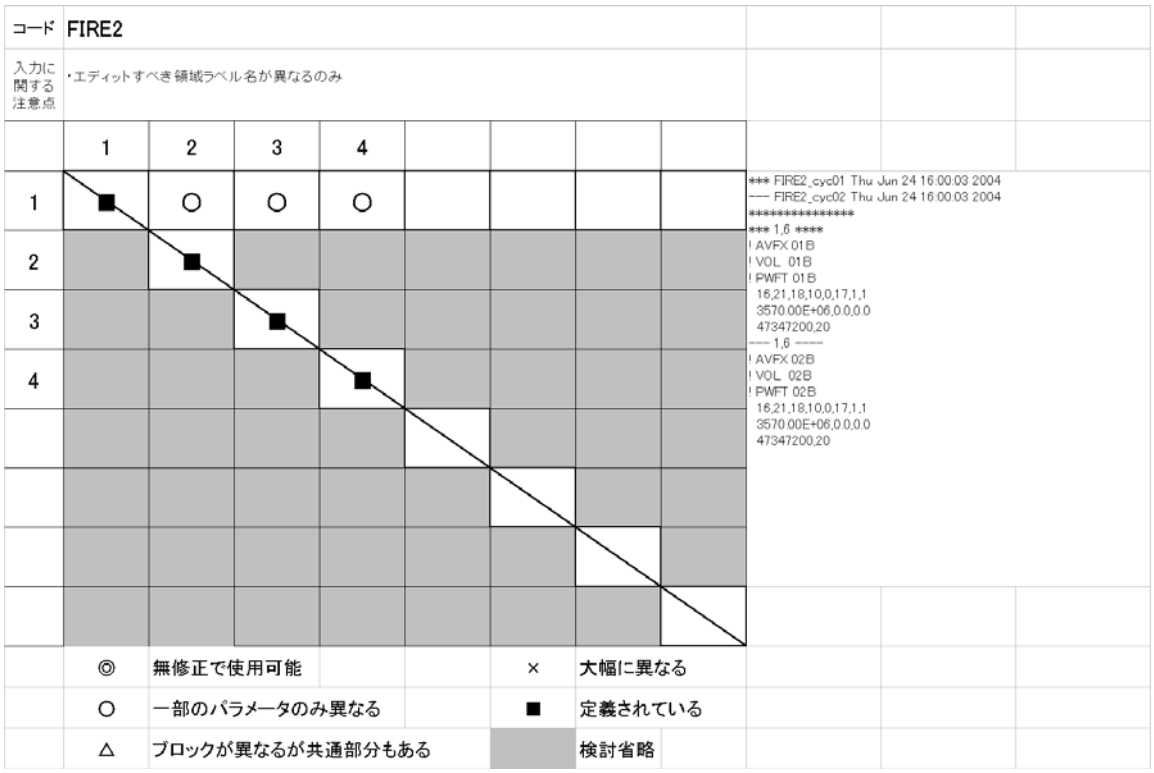


図 2-7　Fire2 コードへの入力ファイルにおける解析ケース間での類似性検討

コード	STARTUP							
入力に関する注意点	・無修正で使用可能							
	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1	■	◎	◎	◎	◎	◎	◎	◎
E-1		■						
B-2			■					
E-2				■				
B-3					■			
E-3						■		
B-4							■	
E-4								■
	◎	無修正で使用可能				×	大幅に異なる	
	○	一部のパラメータのみ異なる				■	定義されている	
	△	ブロックが異なるが共通部分もある					検討省略	

図 2-8　Startup コードへの入力ファイルにおける解析ケース間での類似性検討

コード	NSINI4							
入力に関する注意点	・入力ファイルは一つのみ							
	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1								
E-1								
B-2								
E-2								
B-3								
E-3								
B-4								
E-4								
	◎	無修正で使用可能				×	大幅に異なる	
	○	一部のパラメータのみ異なる				■	定義されている	
	△	ブロックが異なるが共通部分もある					検討省略	

図 2-9　Nsini4 コードへの入力ファイルにおける解析ケース間での類似性検討

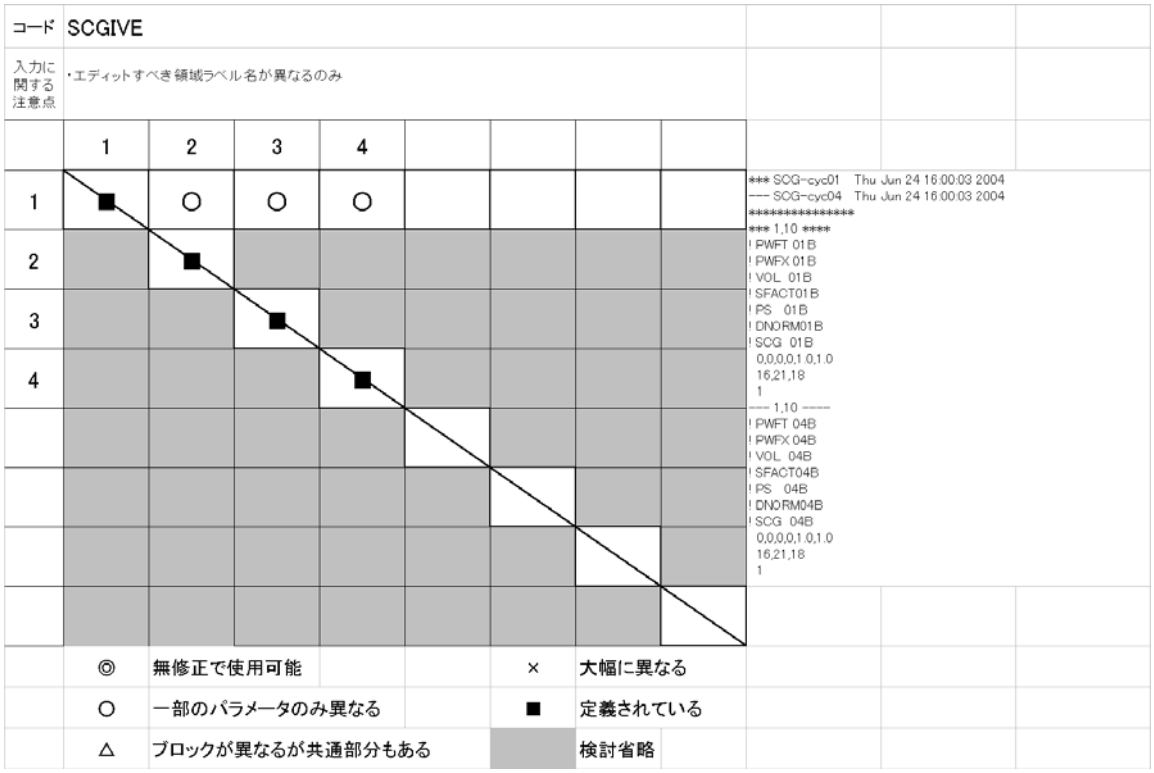


図 2-10　Scgive コードへの入力ファイルにおける解析ケース間での類似性検討

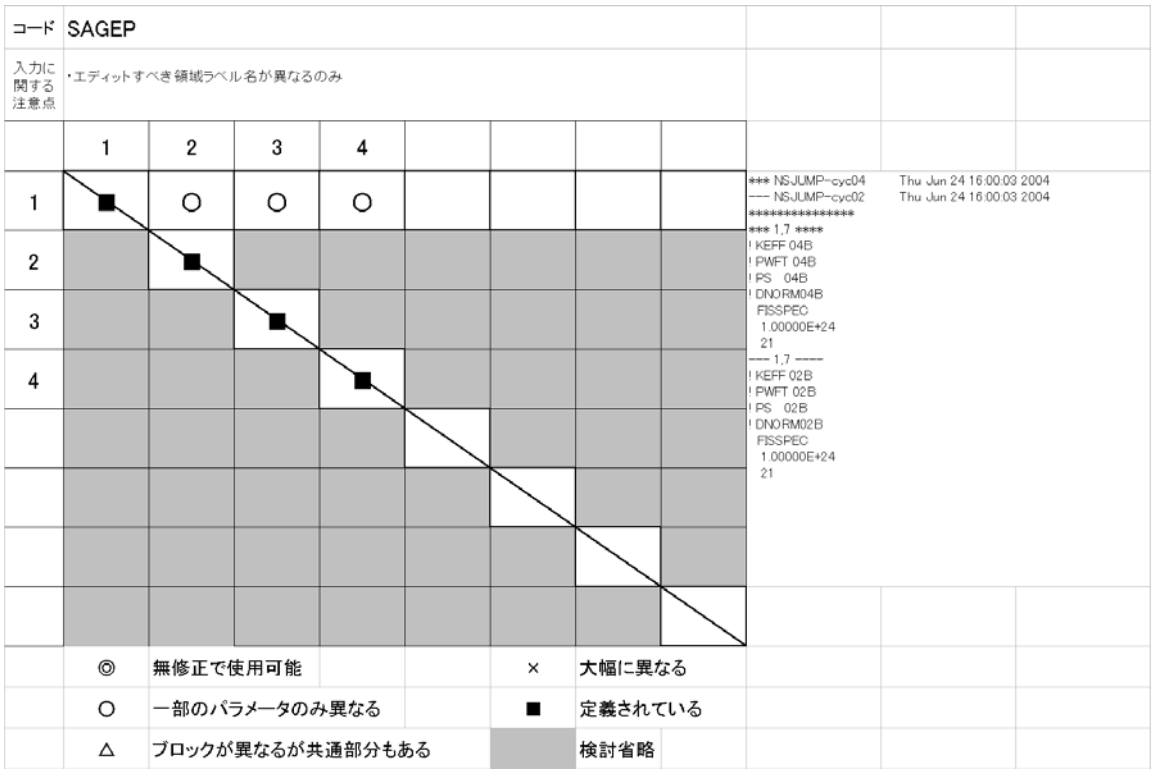


図 2-11　Sagep コードへの入力ファイルにおける解析ケース間での類似性検討

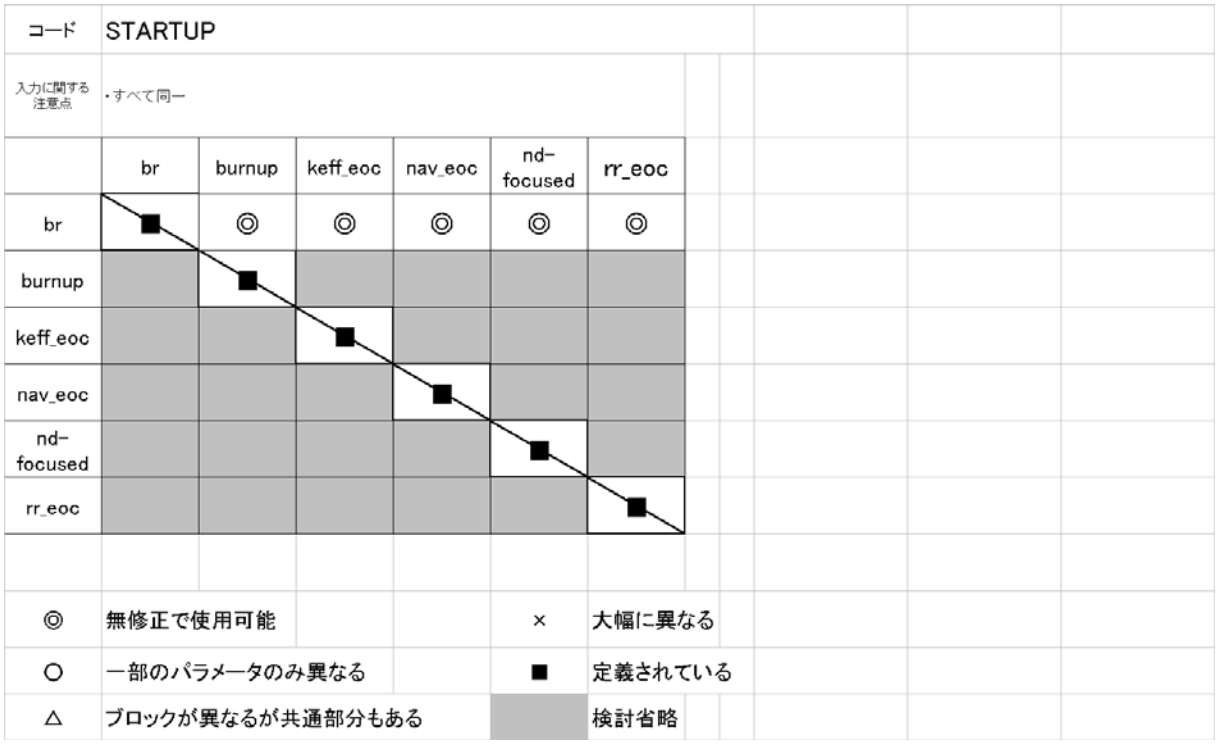
コード	REFUEL2							
入力に関する注意点	・最終サイクルからのサイクル数情報がことなるのみ ・BOC及びEOCでのサイクル間では類似性がある。 ・ただし、Cyc04 EOC のみ大幅に異なる(ブロック追加)							
	B-1	E-1	B-2	E-2	B-3	E-3	B-4	E-4
B-1	■							
E-1		■						
B-2			■					
E-2				■				
B-3					■			
E-3		○		○		■		
B-4		×	○	×	○	×	■	
E-4	×	×	×	×	×	×	×	■
	◎	無修正で使用可能				×	大幅に異なる	
	○	一部のパラメータのみ異なる				■	定義されている	
	△	ブロックが異なるが共通部分もある					検討省略	

*** Refuel2-cyc04BOC Thu Jun 24 16:00:03 2004
--- Refuel2-cyc02BOC Thu Jun 24 16:00:03 2004
****
*** 1 ****
! 1 1
--- 1 ----
! 3 1

図 2-12　Scgive コードへの入力ファイルにおける解析ケース間での類似性検討

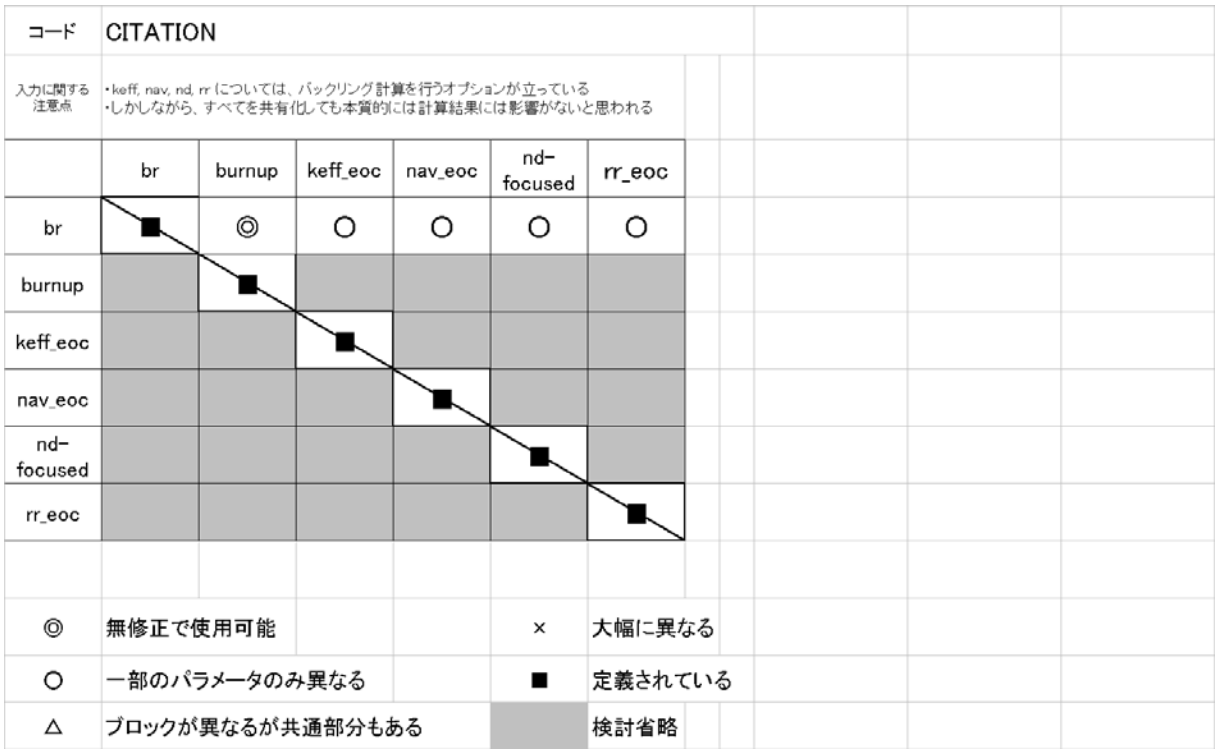
コード	Refuel					
入力に関する注意点	・cyc01-BOC のみ、nd-focused が異なる。					
	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	△	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■
◎	無修正で使用可能				×	大幅に異なる
○	一部のパラメータのみ異なる				■	定義されている
△	ブロックが異なるが共通部分もある					検討省略

図 2-13　Refuel コードへの入力ファイルにおける解析シナリオ間での類似性検討



コード	STARTUP					
入力に関する注意点	・すべて同一					
	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	◎	◎	◎	◎
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■
◎	無修正で使用可能			×	大幅に異なる	
○	一部のパラメータのみ異なる			■	定義されている	
△	ブロックが異なるが共通部分もある				検討省略	

図 2-14　Startup コードへの入力ファイルにおける解析シナリオ間での類似性検討



コード	CITATION					
入力に関する注意点	・keff, nav, nd, rr については、バックリング計算を行うオプションが立っている ・しかしながら、すべてを共有化しても本質的には計算結果には影響がないと思われる					
	br	burnup	keff_eoc	nav_eoc	nd-focused	rr_eoc
br	■	◎	○	○	○	○
burnup		■				
keff_eoc			■			
nav_eoc				■		
nd-focused					■	
rr_eoc						■
◎	無修正で使用可能			×	大幅に異なる	
○	一部のパラメータのみ異なる			■	定義されている	
△	ブロックが異なるが共通部分もある				検討省略	

図 2-15　Citation コードへの入力ファイルにおける解析シナリオ間での類似性検討

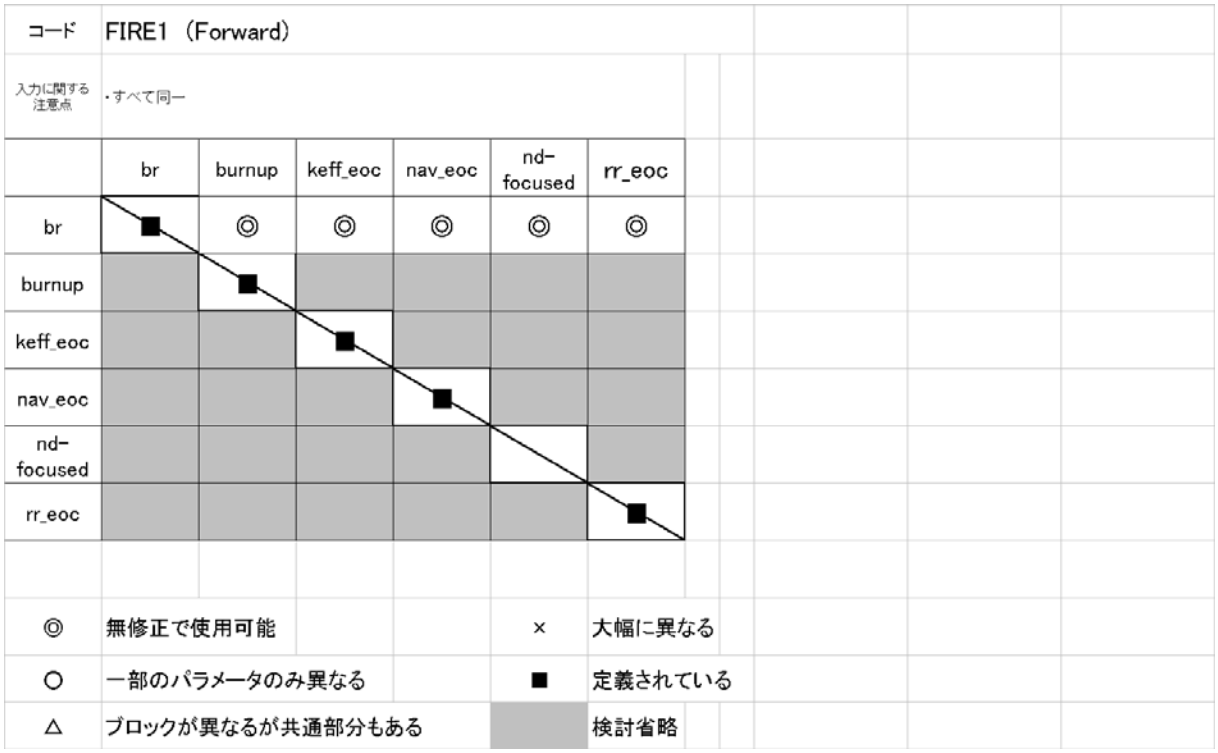


図 2-16　Fire1 コード(通常計算)への入力ファイルにおける解析シナリオ間での類似性検討

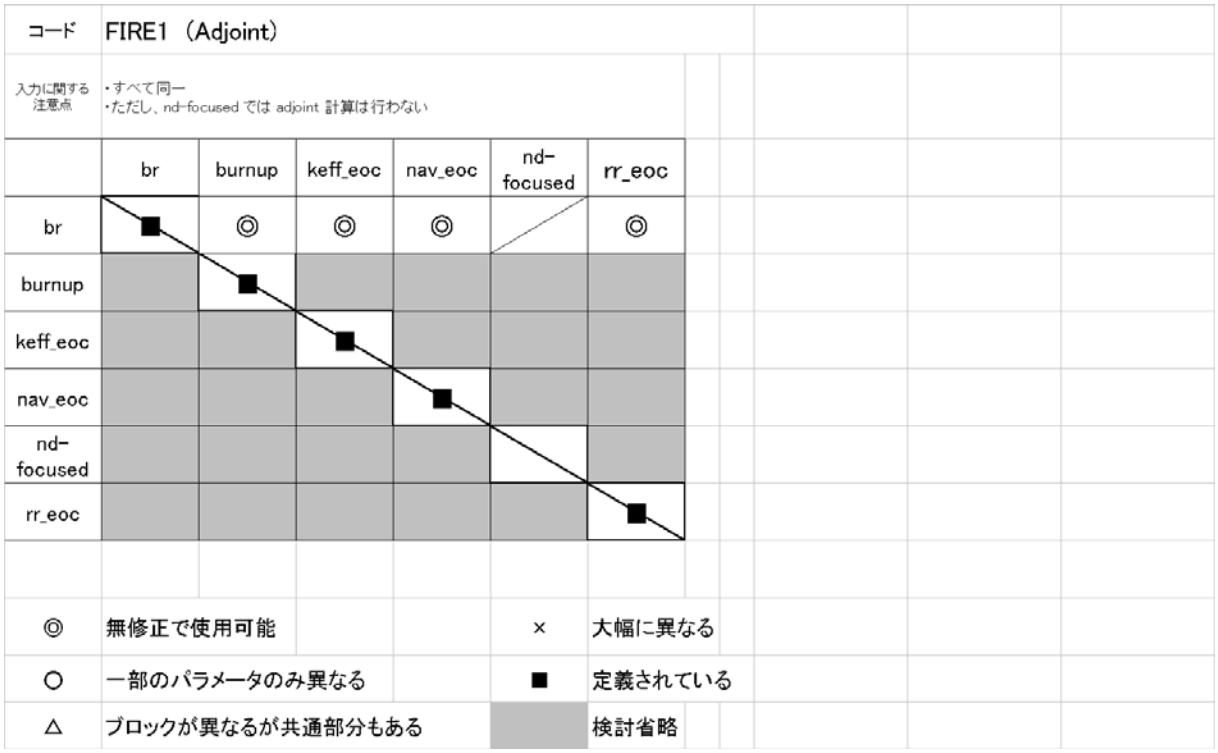


図 2-17　Fire1 コード(随伴計算)への入力ファイルにおける解析シナリオ間での類似性検討

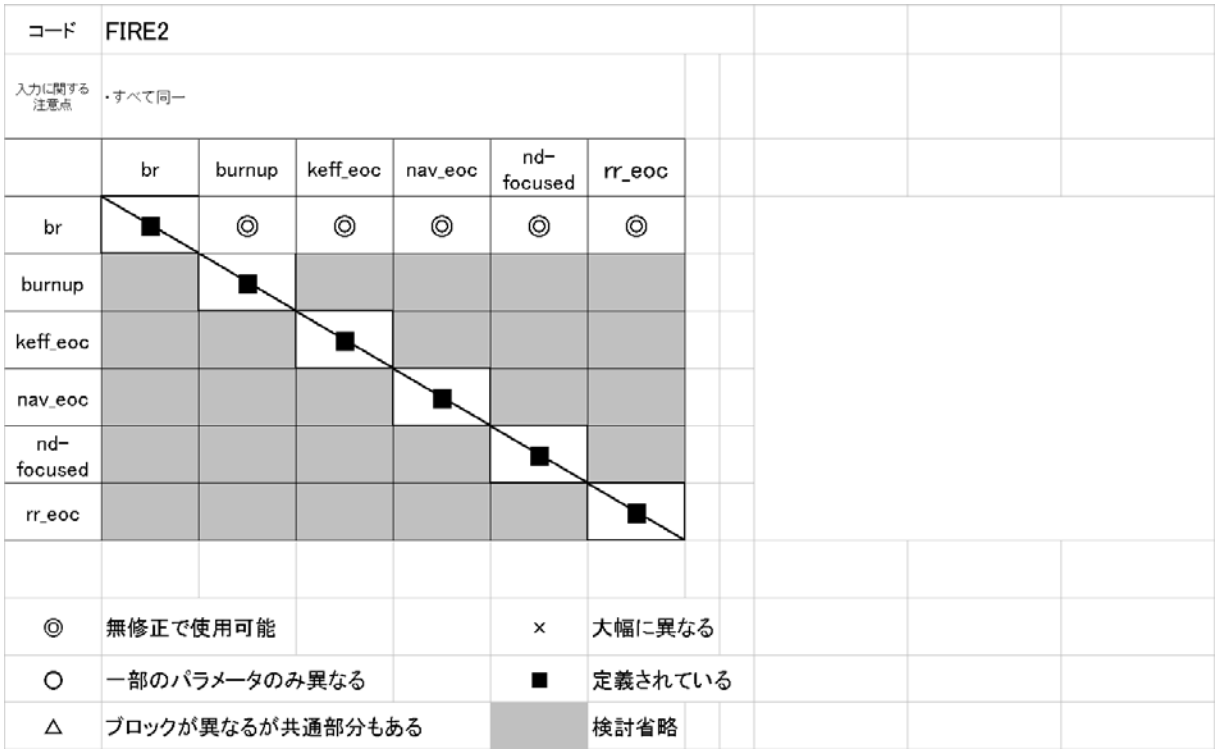


図 2-18　Fire2 コードへの入力ファイルにおける解析シナリオ間での類似性検討

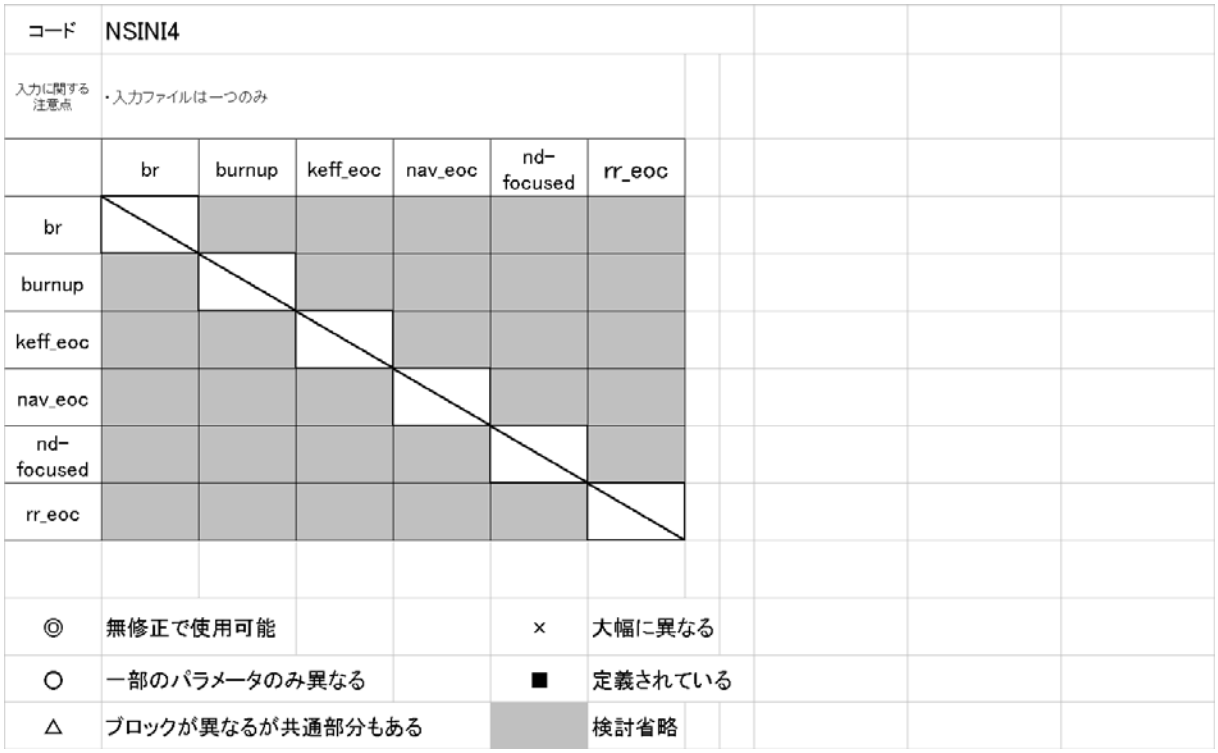


図 2-19　Nsini4 コードへの入力ファイルにおける解析シナリオ間での類似性検討

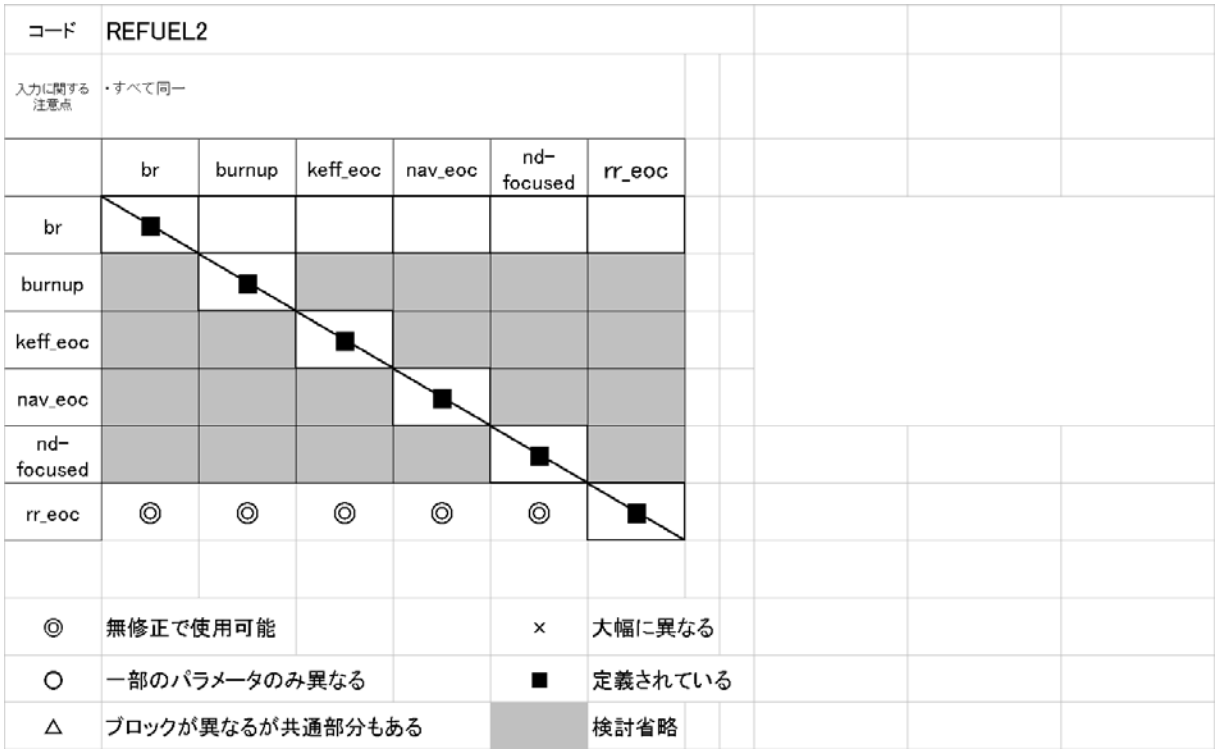


図 2-20　Refuel2 コードへの入力ファイルにおける解析シナリオ間での類似性検討

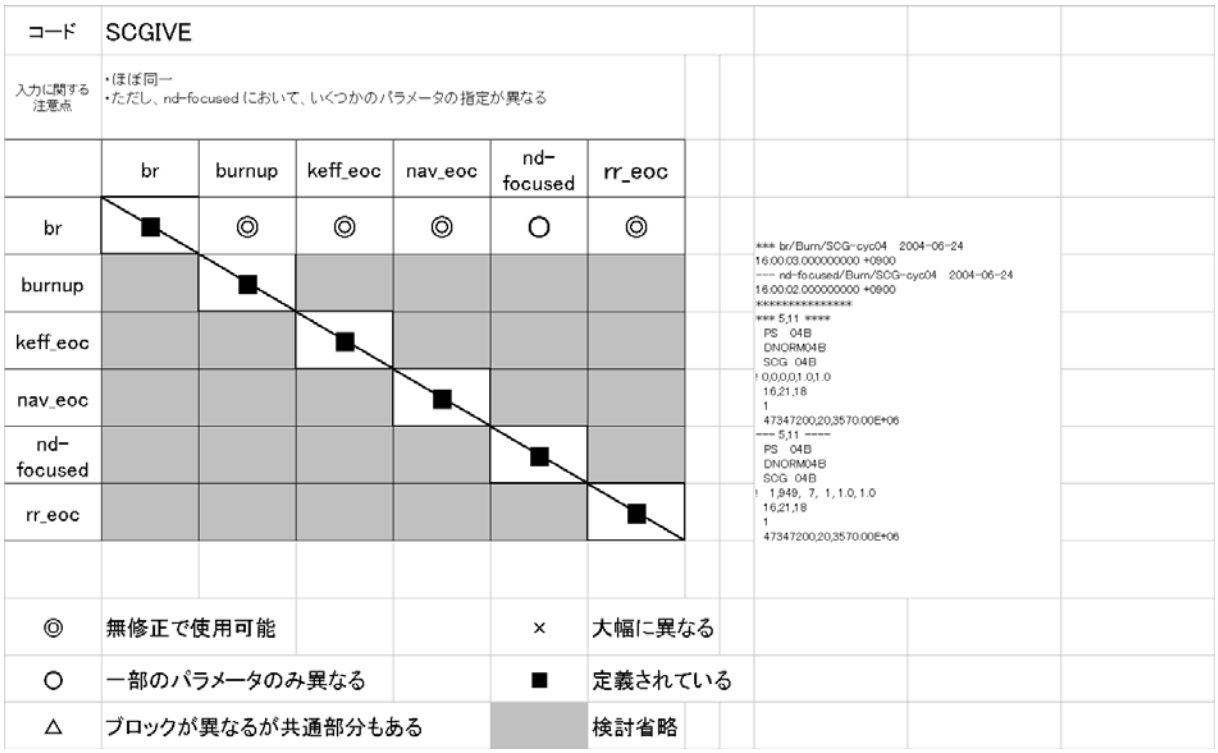


図 2-21　Scgive コードへの入力ファイルにおける解析シナリオ間での類似性検討

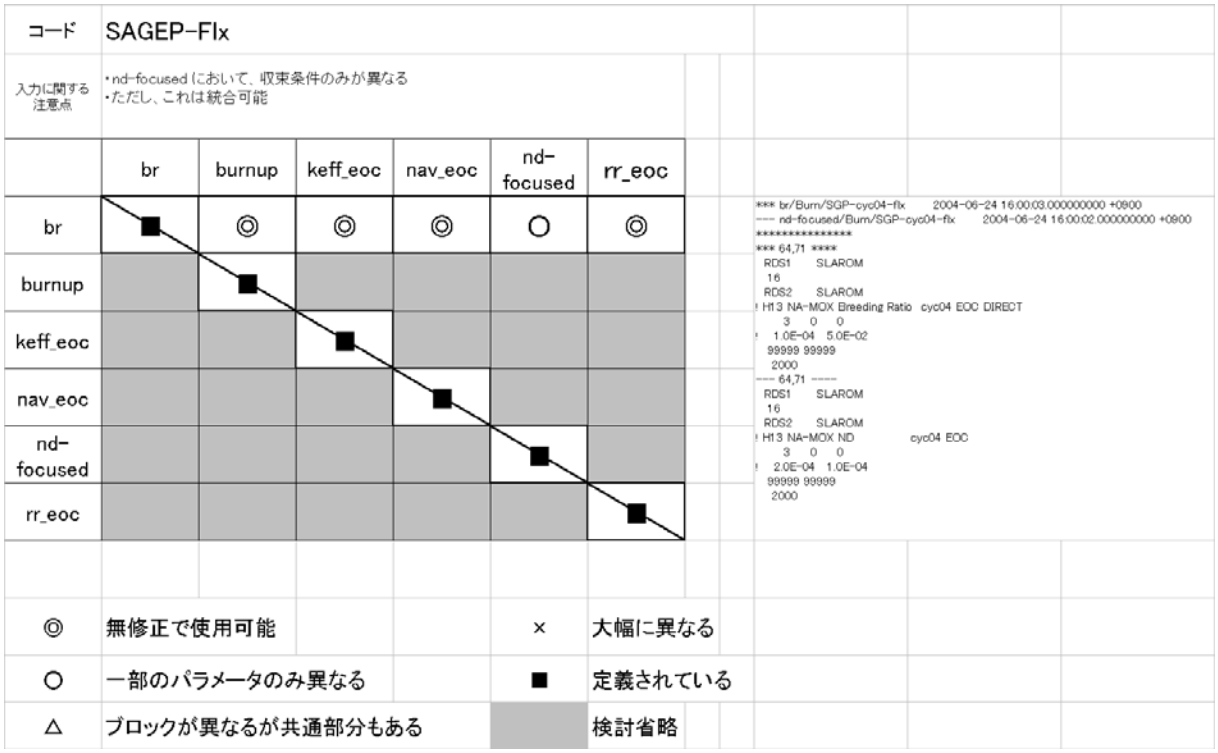


図 2-22　Sagep コード(中性子項計算)への入力ファイルにおける解析シナリオ間での類似性検討

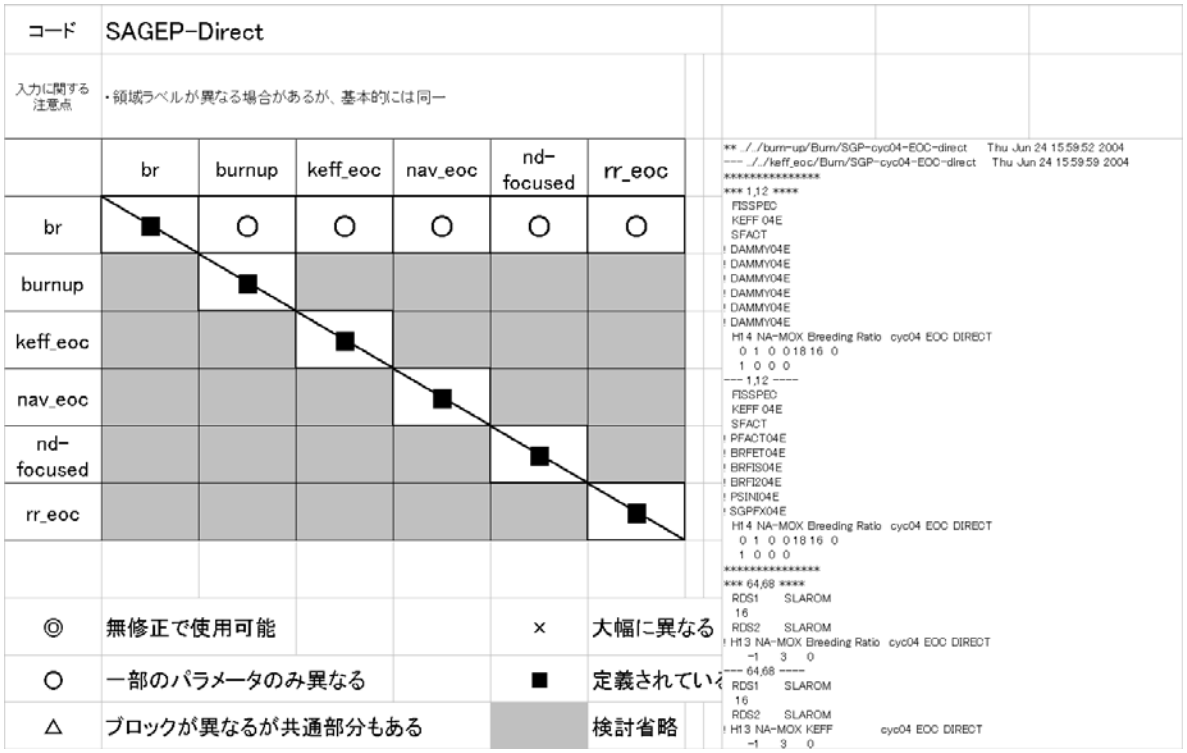


図 2-23　Sagep コード(直接項計算)への入力ファイルにおける解析シナリオ間での類似性検討

3. PSAGEP システムの実装

前章の設計方針に基づき、フェーズ２の実装を行った。本章では、PSAGEP のシステム構成の概要について述べた後、フェーズ１及びフェーズ２で開発した各クラスの機能について説明する。

3.1 モジュールパッケージ

PSAGEP システムでは、標準的な Python システムと同様に、幾つかの機能の単位に分割して分割され、モジュールパッケージとして管理されている。モジュールは、図 3-1 に示すように階層的に管理されている。

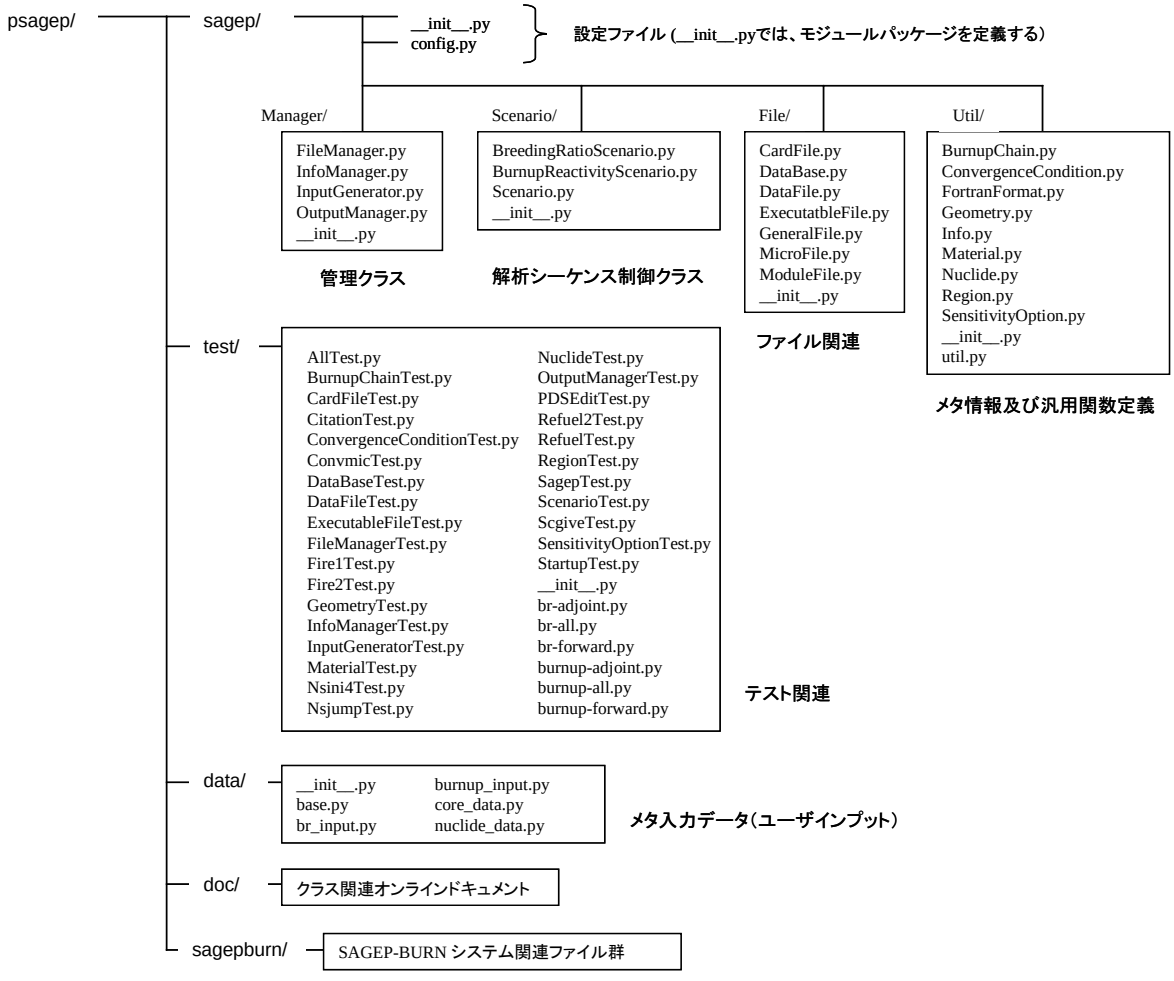


図 3-1　PSAGEP システムのモジュールパッケージ

なお、モジュールの階層関係は、インストールされた後のディレクトリ構造と一致している。モジュールは、表 3-1 に示すように、幾つかの機能毎に分割して管理されている。

表 3-1　モジュールパッケージ名とその機能

パッケージ名 / サブパッケージ		役割
sagep	sagep.Manager	管理クラス
	sagep.Scenario	解析シーケンス制御クラス
	sagep.File	ファイル関連クラス
	sagep.Util	メタ情報クラス及び汎用関数
data		解析のための入力メタ情報定義
test		テスト関連クラス

これらのパッケージを利用するためには、Python を利用するに当たっての設定が必要である。（詳細は第 4 章を参照）

3.2 各クラスの概要

各モジュールパッケージは、幾つかのクラス定義ファイルにより構成されている。PSAGEP を構成する各クラス間の関係を図 3-2 に示す。なお、図において灰色で示されている部分については、フェーズ 2 にて開発された部分を表している。

以下では、パッケージ毎に各クラスの概要について述べる。

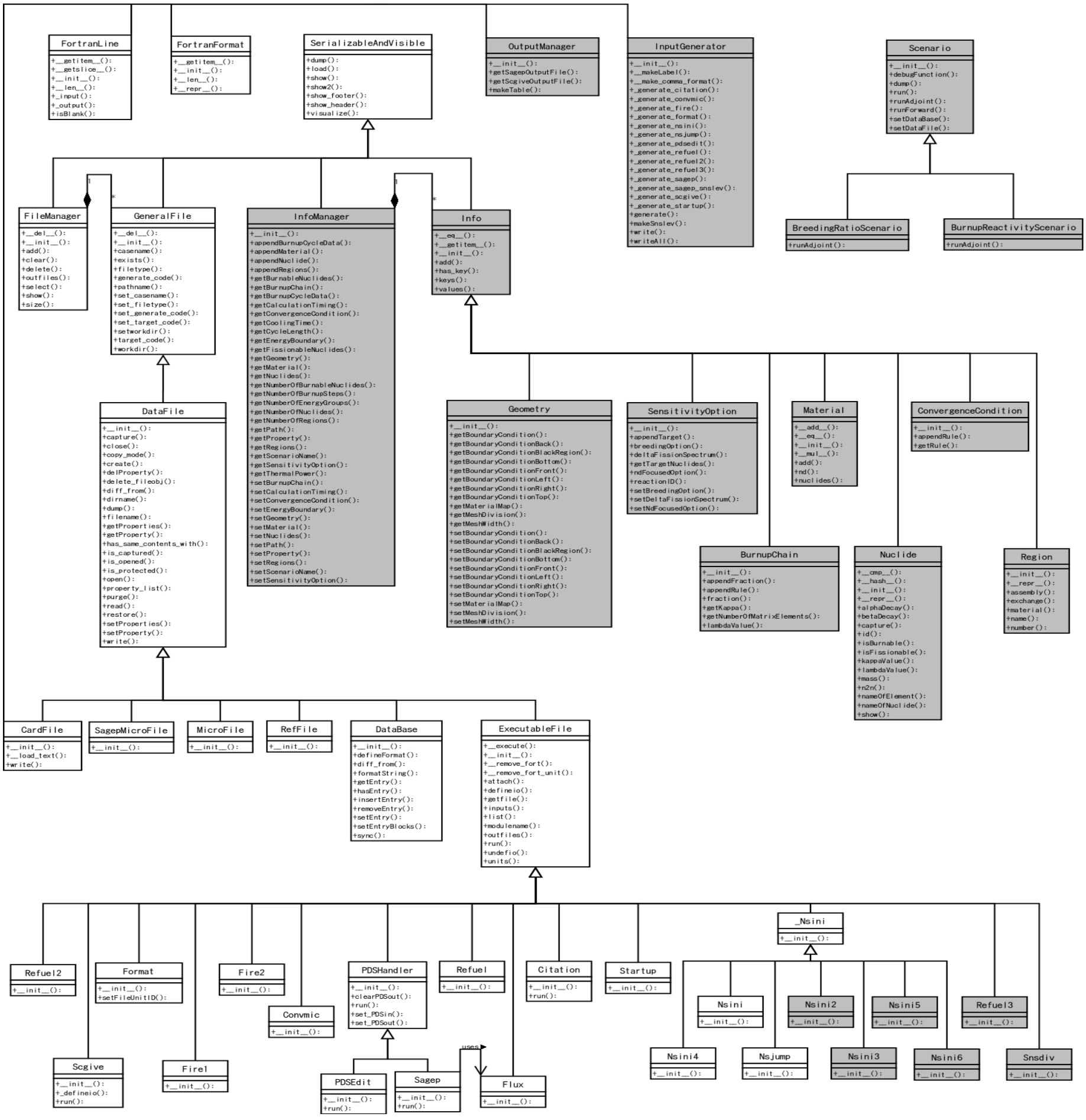


図 3-2 PSAGEP システムを構成するクラス群（クラス図）

3.2.1 管理クラス

管理クラスは、システムの機能単位を司るものである。表 3-2 に管理クラスの一覧を示し、以下に各クラスの概要を示す。

- FileManager クラスは、ExecutableFile クラスによって出力されたファイルを保持する、一種のコンテナとなっている。GeneralFile クラスで定義されている属性（生成コード、ファイルタイプ、ケース名）を検索条件として、対象となるファイルオブジェクトの抽出を行う。
- InfoManager クラスは、入力メタデータである Info オブジェクトを管理するものであり、各種の情報を設定・取得することができる。
- InputGenerator クラスは、InfoManager クラスからメタ情報を取得し、バックエンドの計算コードのための入力ファイルを自動的に生成する。
- OutputManager クラスは、FileManager クラスからバックエンドの計算コードが生成したアウトプットファイルから、感度係数情報を抽出し、ユーザが利用しやすい形に結果を整理する。

3.2.2 解析シーケンス（シナリオ）制御クラス

解析シーケンス制御クラスは、シナリオクラスとも呼ばれ、着目する核特性毎に異なるクラスとして準備される。表 3-3 に、解析シナリオ制御クラスの一覧を示す。

同クラスにおいては、バックエンドの計算コードを用いた、一連の解析シーケンスが記述されている。従って、解析シーケンス（解析シナリオ）を新設する場合には、各計算コードを用いた解析方法についてある程度理解しておくことが必要となる。ただし、計算コードのための入力ファイルやコード間のファイル接続については、InputGenerator や各コードのラッパー定義情報を利用してシステムが自動的に解決するため、従来に比べて非常に容易になっている。以下に各クラスの概要を示す。

- Scenario クラスは各シナリオの基底クラスである。解析シナリオに依存しない Forward 計算シーケンスと、一部の共通メソッドが定義されている。
- BreedingRatioScenario と BurnupReactivitySenario は、それぞれ増殖比と燃焼反応度損失の燃焼感度解析を求めるクラスであり、Adjoint 計算シーケンスが定義されている。

3.2.3 ファイル関連クラス

ファイル関連クラスには、図 3-2 に示すように、継承関係がある。以下、各クラスの概要を示す。表 3-4 に、ファイル関連クラスの一覧を示す。

- GeneralFile がファイル関連クラスの基底クラスであり、基本的なメソッドが定義され居ている。GeneralFile クラスは、システム上に存在する「ファイル」へのポインタ情報のみを管理している。
- DataFile は、GeneraFile の機能に加えて、バックエンド計算コード向けの入力ファイルや、計算コードから出力されたのデータファイル等の、「ファイル」をオブジェクト化したものである。DataFile はファイル内容の読み込みや書き込み、ファイル間の内容比較・差分情報作成等が可能である。
- CardFile は、固定フォーマットのファイル作成に特化したクラスであり、主としてバックエンド計算コード向けの入力ファイル作成時に利用する。
- ExecutableFile は、各バックエンド計算コードのラッパーの基底クラスである。Fortran I/O ユニットとのファイル接続に関する依存関係の情報の設定、ファイル依存関係の自動解決、計算コードの実行等を行う。
- その他クラスは ModuleFile.py で定義されてそれぞれに対応したファイル入出力情報が定義されており、この情報を元にファイル接続の依存関係が自動的に解決される。

3.2.4 メタ情報およびその他

メタ情報クラスは、ユーザが指定するメタ情報を保持するコンテナオブジェクトである。表 3-5 に、メタ情報およびその他クラスの一覧を示す。なお、表にはメタ情報を定義する上で必要となるクラスも含んでいる。これらは見かけ上 Info クラスを継承しているが、特に共通するメソッドがあるというわけではなく、ここでは階層関係を示すために導入した。以下に、各クラスの概要を示す。

- Nuclide クラスは、核種を抽象データオブジェクト化したものであり、JFS セット ID や物性値情報等を管理する。
- Geometry クラスは、バックエンド計算コードに与えるための体系情報（メッシュ幅、メッシュ数）や、境界条件、マテリアルマップの各情報を保持している。
- Mateial クラスは、核種毎の数密度情報を保持している。また、Material オブジェクトに関する演算（Material 間の和、スカラー値との積）にも対応しており、マテリアルの重み付け平均等を実現する。
- Region クラスは、マテリアルと領域名を結びつけた情報を保持している。
- BurnupChain クラスは、燃焼チェーン情報を保持する。燃焼チェーンの設定には、親核種、娘核種、フラクションの 3 情報を用いて容易に設定することが出来る。指定されたルールセットから、バックエンド計算コード向けの各燃焼マトリクスを自動的に生成する。

- ConvergenceCondition クラスは、バックエンド計算コード(SAGEP および CITATION)による中性子束計算の収束判定条件を保持する。
- SensitivityOption クラスは、バックエンド計算コード(SAGEP および SCGIVE)に指定する、対象とする反応タイプ等の解析のための入力情報を保持する。

3.2.5 テスト関連クラス

テスト関連クラスは、ユニットテスト向けクラスと、結合テスト向けクラスに分類できる。前者は、当該クラスを対象とした、いわゆる単体テストを行うものであり、メソッドの挙動についてテストを行っている。また、後者については、クラス間の関連や挙動について、その結合性について検証している。実際には、実際の解析シーケンスに基き、バックエンド計算コードのラッパーオブジェクトを操作して一連の計算を実施し、参照解と一致することを確認している。

表 3-2　管理関連の各クラスの機能

クラス名	機能
FileManager	データファイルの入出力管理
InfoManager	メタ入力情報の管理
InputGenerator	メタ入力情報から計算コード向け入力ファイルの生成
OutputManager	出力ファイルから感度係数テーブルの生成

灰色部はフェーズ 2 で実装したものを示す(以下同様)

表 3-3　解析シーケンス（シナリオ）制御関連の各クラスの機能

クラス名	機能
Scenario	解析シナリオに関する基底クラス
BreedingRatioScenario	増殖比に関する解析シナリオクラス
BurnupReactivitySenario	燃焼反応度損失に関する解析シナリオクラス

※クラス名のインデントは、継承関係を示す

表 3-4　ファイル関連の各クラスの機能

クラス名	機能
GeneralFile	ファイル操作に関する基底クラス
DataFile	データファイルの取り扱い
CardFile	ユーザカード(fort.5)の取り扱い
ExecutableFile	ロードモジュール・ラッパーのための基底クラス
Convmic	Convmic モジュールのラッパー
Citation	Citation モジュールのラッパー
Fire1	Fire1 モジュールのラッパー
Fire2	Fire2 モジュールのラッパー
PDSHandler	PDS を取り扱うモジュールのための基底クラス
PDSEdit	PDSEdit モジュールのラッパー
Sagep	Sagep93 モジュールのラッパー
Flux	Flux モジュールのラッパー
_Nsini	Ns 系モジュールのための基底クラス
Nsini	Nsini モジュールのラッパー
Nsini2	Nsini2 モジュールのラッパー
Nsini3	Nsini3 モジュールのラッパー
Nsini4	Nsini4 モジュールのラッパー
Nsini5	Nsini5 モジュールのラッパー
Nsini6	Nsini6 モジュールのラッパー
Nsjump	Nsjump モジュールのラッパー
Refuel	Refuel モジュールのラッパー
Refuel2	Refuel2 モジュールのラッパー
Refuel3	Refuel3 モジュールのラッパー
Snsdiv	Snsdiv モジュールのラッパー
Scgive	Scgive(scgivebr)モジュールのラッパー
Startup	Startup モジュールのラッパー

※クラス名のインデントは、継承関係を示す

表 3-5　メタ情報およびその他に関する各クラスの機能

クラス名	機能
Info	メタ情報の基底クラス
Geometry	体系情報を管理するクラス
Nuclide	核種情報を管理するクラス
Material	物質情報を管理するクラス
Region	領域情報（交換集合体等）を管理するクラス
BurnupChain	燃焼チェーン情報を管理するクラス
ConvergenceCondition	収束条件を管理するクラス
SensitivityOption	解析対象の核種等の情報を管理するクラス
SerializableAndVisible	オブジェクトの内部状態の表示
PSagep	Psagep システム全体の制御クラス
FortranFormat, FortranLine	Fotran形式書式によるプレーンテキストの読み書き (ScientificPython[10]のものを機能拡張)

※クラス名のインデントは、継承関係を示す。灰色部はフェーズ 2 で実装したものを示す(以下同様)

表 3-6　ユニットテスト関連のスクリプトファイル名とテスト対象クラス

ファイル名	テスト対象クラス
AllTest.py	下記のテストをすべて実行
BurnupChainTest.py	BurnupChain
CardFileTest.py	CardFile
CitationTest.py	Citation
ConvergenceConditionTest.py	ConvergenceCondition
ConvmicTest.py	Convmic
DataBaseTest.py	DataBase
DataFileTest.py	DataFile
ExecutableFileTest.py	ExecutableFile
Fire1Test.py	Fire1
Fire2Test.py	Fire2
GeometryTest.py	Geometry
InfoManagerTest.py	InfoManager
InputGeneratorTest.py	InputGenerator
MaterialTest.py	Material
Nsini4Test.py	Nsini4
NsjumpTest.py	Nsjump
NuclideTest.py	Nuclide
OutputManagerTest.py	OutputManager
PDSEditTest.py	PDSEdit
Refuel2Test.py	Refuel2
RefuelTest.py	Refuel
SagepTest.py	Sagep
ScgiveTest.py	Scgive
SensitivityOptionTest.py	SensitivityOption
StartupTest.py	Startup

表 3-7　結合テスト関連のスクリプトファイル名とテスト内容

ファイル名	テスト内容
br-forward.py	増殖比に関する Forward 計算
br-adjoint.py	増殖比に関する Adjoint 計算
br-all.py	増殖比に関する Forward 及び Adjoint 計算
burnup-forward.py	燃焼反応度損失に関する Forward 計算
burnup-adjoint.py	燃焼反応度損失に関する Adjoint 計算
burnup-all.py	燃焼反応度損失に関する Forward 及び Adjoint 計算
ScenarioTest.py	解析シナリオ(Scenario)を用いた総合機能テスト

3.3 検証

2.1 節において述べたように、本システムは「テストファーストプログラミング」(Test First Programming)の考え方に基づき開発を行った。すなわち、まずテスト対象のクラスオブジェクトの挙動を定義したテストコードを記述し、そのテストが通るように PSAGEP システムの実装をおこなった。従って、3.2.5 節で示したように、テストコードは、すべてのクラスに対して用意されている。なお、これらのテストコードの記述にはテスティングフレームワークである PyUnit を使用した。

テストコードは、ユニットテスト及び結合テストの２種類に分類される。前者は、いわゆる単体テストであり、当該クラスにおける各メソッドの挙動をテストするものである。後者は、ファイル受け渡し等のオブジェクト間の関係をテストする物である。

これらのテストにおいて、動作確認用データとして、実施済みである増殖比に関する燃焼感度解析における入力ファイル、出力ファイル、計算リストを利用した。

3.3.1 ユニットテスト

表 3-3 にユニットテストの一覧を示す。各ユニットテストのためのクラスは、テスト対象クラスの名前の後に”Test”が追加されている。バックエンド計算コードのラッパークラスのテストの際には､入出力ファイル及び計算リストが参照計算のものと一致することを確認した。

3.3.2 結合テスト

表 3-4 に結合テストの一覧を示す。これらは、増殖比および燃焼反応度損失の感度解析のための Forward 計算及び Adjoint 計算に関するものであり、解析シナリオにおけるファイルの接続関係をテストしている。また、ScenarioTest は、PSAGEP システムの全てのモジュールを用いた総合テストとなっている｡これらのテストにより得られた計算結果は参照計算のものと完全に一致することを確認し、本システムの妥当性を確認した。

4. PSAGEP システムを用いた解析

本章では、PSAGEP システムを用いた解析の手順に関して、実際の例を通じて説明する。

4.1 解析の為の準備

本節では、PSAGEP システムを新規に導入する際に必要な手順について示す。なお、Python 言語のバージョン 2.3 以降のものがインストールされていることを仮定している。

4.1.1　インストール手順

CD-ROM には以下のファイルが含まれる。

- psagep.tgz

適当なディレクトリに psagep をインストールする。ここでは、“/usr/local/”にインストールした場合を例として説明する。

```
# cd /usr/local/
# cp {CD-ROM}/psagep.tgz ./psagep.tgz
# tar zxvf psagep.tgz
```

tar zxvf {ファイル名}.tgz は

```
# gunzip {ファイル名}.tgz
# tar xvf {ファイル名}.tar
```

で代用しても良い。次に（必要であれば）PSAGEP システムの環境設定を行う。環境設定を行うファイルは/usr/local/psagep/sagep/config.py である。インストールするディレクトリが/usr/local と異なる場合、source_dir の値を/usr/local/psagep/から変更する。

続いて、SAGEP-BURN 実行モジュールのコンパイルを行う。

```
# cd /usr/local/psagep/sagepburn/CODE/sagep-burn/src
# csh MAKE
# cd /usr/local/psagep/sagepburn/CODE
# csh mklall.sh
```

コンパイルが正常に終了すれば、PSAGEP のインストールは終了である。

4.1.2 環境設定

次にユーザ環境の設定を行う。適当なエディタを用いて.cshrc を開き、PATH、PYTHONPATH の設定を行う。

- .cshrc に以下を追加

  set path=(/usr/local/psagep $path)

  setenv PYTHONPATH “/usr/local/psagep:/usr/local/lib/python2.3”

- 以下のコマンドを実行し、ユーザ環境設定を行う。

```
# source .cshrc　（変更の反映）
# which psagep.py　（パスの確認）
/usr/local/psagep/psagep.py　←　このように表示されれば OK
>> echo $PYTHONPATH　（Python パスの確認）
/usr/local/psagep:/usr/local/lib/python2.3
```

4.2 解析の概要

前節では、PSAGEP システムを使用する為の環境整備を行った。本節では、PSAGEP システムを用いて一連の解析を行う手順について説明する。

4.2.1 シナリオについて

燃焼感度解析を行う際には、解析対象とする感度ごとに解析シーケンスを変更する必要がある。また、解析サイクル数によっても解析シーケンスは異なる。ここで、解析シーケンスは解析条件によって多少変化するものの、基本的な形は変わらない。この解析シーケンスを汎化させたものをシナリオと呼ぶ。

4.2.2 出力されるファイルについて

計算が最後まで終了すると、SNS_{解析シナリオ名}というファイルができる。このファイルが PSAGEP の最終結果である。

また、Forward 計算が終了すると、Forward 計算中の各計算で作成された断面積、中性子束等のリスタートファイルを全て格納した{シナリオ名}-Forward.dat というファイルが作成される。同様に Adjoint 計算が終了すると、{シナリオ名}-Adjoint.dat というファイルが作成される。また、データベースファイルは逐一、DataBase-Forward-{解析シナリオ名}という名前で書き出される。（Adjoint 計算時には Forward が Adjoint になる）

解析シナリオが変わったとしても、Forward 計算の内容は同一であるため、Forward 計算のリスタートファイル、データベースファイルは複数の解析シナリオで共用することが可能である。

4.2.3 中間生成物について

psagep を実行すると、config.py で指定した tmp_dir の下に、psagep-{ユーザ名}-{プロセス ID}というディレクトリが作成される。計算は全てこのディレクトリで実行される。リスタートファイル、最終結果（SNS ファイル）以外の中間生成物、SAGEP-BURN の各計算に必要な入力、出力などは全てこのワークディレクトリ内に生成される。

4.2.4 テスト用リファレンス結果の作成

テストコードでは、従来 SAGEP-BURN システムの各計算コードのインプット、及び解析結果と、PSAGEP システムによって自動生成された計算コードのインプット、解析結果の比較を行う。

このため、テストコードの実行を行う際には、リファレンスとなる従来 SAGEP-BURN システムの解析結果が存在している必要がある。従来システムの実行は以下のように行う。ここでは増殖比を例に挙げて説明する。

```
# cd /usr/local/psagep/sagepburn/br/Burn
# sh Burn-normal.sh
# sh Burn-adj.sh
```

計算が正しく終了すると、各計算コードに対して、{インプットファイル名}.dmp というファイルができているはずである。次に、計算結果の処理を行う。

```
# cd ../table
# sh mk_shs.sh
```

これが終了すると、SNS というファイルができる。これは、PSAGEP の最終生成物と、等価のものである。

4.3 ユーザ入力の作成

PSAGEP の入力は大きく以下の 3 つに分けられる。

1. エネルギー群構造、核種名、燃焼チェーン等の核種に関係するデータ（nuclide_data.py）
2. 炉心体系、領域に含まれるマテリアル等の炉心構造に関係するデータ（core_data.py）
3. 燃焼ステップ、収束条件、感度計算オプション等のユーザが頻繁に変更するデータ（br_input.py 等）

これらのデータは /usr/local/psagep/work/psagep/data に存在する。なお、入力ファイルの詳細については、付録 1 を参照のこと。

以下では、ユーザ入力を作成するために必要な文法の説明、及びサンプルインプットを例にとって、ユーザーインプットの各パラメータについての説明を行う。

4.3.1 Python の文法の説明

PSAGEP のユーザ入力は全て Python 形式で記述される。ここではまず、入力を記述するのに最低限必要な Python の文法を次に示す。

(1) 変数の代入

Python はスクリプト言語なので、宣言文なしで変数を用いることができる。型宣言無しで

変数 = 式

として問題ない。

(2) リスト

リストとは一般に配列と呼ばれるものである。Python のリストは、コンマで区切られた 値からなるリストを各カッコで囲んだものとして書き表される。リストの要素をすべて同じ型にする必要はない。また、リストの引数は 0 から始まる。

```
% python
Python 2.3.2 (#1, Dec  4 2003, 11:29:08)
[GCC 2.96 20000731 (Red Hat Linux 7.3 2.96-113)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> A = [ 'a', 'b', 10, 100 ]
>>> print A
[ 'a', 'b', 10, 100 ]
>>> print A[1], A[2]
'b', 10
>>> print A[0]
'a'
```

(3) ディクショナリ

ディクショナリとは連想配列と呼ばれるものである。ディクショナリは順序付けのされていない キー(key) : 値(value) のペアからなり、キーが (辞書の中で) 一意でなければならない。

```
>>> B = {'a' : 1, 'b' : 2 }
>>> print B
{'a' : 1, 'b' : 2 }
>>> print B['a']
1
```

(4) 文字列

文字列は‘（シングルクォート）で囲まれた部分が文字列として取り扱われる。

(5) コメント

Python では#から行末（改行コード）までがコメントになる。

(6) メソッド

メソッドとは C 言語の関数と同様である。psagep のユーザ入力では、メソッドを用いて入力データセットにデータを追加、若しくは代入（上書）という処理を行う。PSAGEP におけるメソッド名の決定ルールとして、append で始まるメソッドではデータの追加を行い、set で始まるメソッドではデータの上書きを行うことになっている。

4.3.2 nuclide_data.py

このファイルには、核種に関するデータが含まれる。具体的には、以下のものが該当する。

- エネルギー群構造
- SAGEP-BURN で用いる核種名
- 燃焼チェーン
- 核種毎の感度計算を行う反応タイプ

以下ではエネルギー群構造から順に、入力形式を説明していく。

(1) エネルギー群構造

エネルギー群構造はリスト形式で入力する。下表に 18 群のエネルギー群構造を示す。これからわかるように、計算対象とするエネルギー範囲の上限と下限を入力する為、18 群構造の場合、入力するデータ数は 19 個となる。

```
### Energy group Info.
jfs18g =   [  1.00000000e+07,    6.06530700e+06,    3.67879400e+06,    2.23130200e+06,
              1.35335300e+06,    8.20850000e+05,    3.87742094e+05,    1.83156406e+05,
              8.65169531e+04,    4.08677109e+04,    1.93045391e+04,    9.11882031e+03,
              4.30742480e+03,    2.03468396e+03,    9.61116516e+02,    4.53999298e+02,
              2.14454102e+02,    1.01300903e+02,    9.99999975e-06]
info_man.setEnergyBoundary( jfs18g )
```

図 4-1　18 群エネルギー群構造の入力例

(2)　SAGEP-BURN で用いる核種名

核種名はリスト形式で入力する。核種名を入力時する際に制限事項となるのは、「ハイフンが必要」であることのみである。（例：u-235, U-235, AM-242M,　u-235fp,　PU-235fp ）

取り扱える核種名の一覧は Nuclide クラス（psagep/sagep/Util/Nuclide.py）を参照のこと。

なお、JOINT システムにて作成したミクロ断面積ファイルとの整合性を保つために、ミクロ断面積ファイルに格納されている核種の順番と、InfoManager に指定する核種の順番が一致するように指定する必要がある。また、核種数についても、両者で整合させる必要がある。

```
### Nuclide Set
# The order of nuclides must be consistent with that in the micro XS file generated with JOINT
nuclide_names = [ 'U-235',  'U-236',  'Np-237',  'U-238',   'Np-239',  'Pu-238',   'Pu-239',
                  'Pu-240', 'Pu-241', 'Pu-242',  'Am-241',  'Am-242m', 'Am-243',   'Cm-242',
                  'Cm-243', 'Cm-244', 'Cm-245',  'U-235FP', 'U-238FP', 'Pu-239FP', 'Pu-241FP',
                  'O-16',   'Na-23',  'Cr-nat.', 'Mn-55',   'Fe-nat.', 'Ni-nat.',  'Mo-nat.',
                  'Nd-143', 'W-nat.', 'B-10',    'B-11',    'C-12' ]
nuclide_set = [    Nuclide(name=x) for x in nuclide_names ]

info_man.setNuclides( nuclide_set )

#print nuclide_set
```

図 4-2　核種名の入力例

(3) 燃焼チェーン

燃焼チェーンは appendRule メソッドを用いて定義する。入力形式は

・ appendRule（親核種名，反応タイプ１，反応タイプ２，・・・）

のように、第１引数に親核種名、第２引数以降に反応タイプを入力する。反応タイプの入力形式には文字列とリストの２通りがある。それぞれの使用法について下図の燃焼チェーンの入力例を用いて説明する。

反応タイプの入力は、基本的にはリスト形式で［反応タイプ、娘核種、反応割合］と入力する。ここで、反応割合が１の場合には、反応割合の入力を省略することができる。また、反応タイプから娘核種が一意に決定される場合、例えば U-235 の捕獲反応で U-236 が生成される場合には、娘核種の入力を省略することができる。娘核種を省略する場合には、リストではなく文字列で入力する。注意点として、娘核種の核種名が PSAGEP に登録されていない場合（前項で登録していない場合）、エラーとなる。

```
### Burnup Chain
bc = BurnupChain()
#            Fertile Name, Reaction Type, [ Reaction Type, Daughter], [ Reaction Type, Daughter, Fraction ]
#            If Reactin Type is Fission, you need to input Daughter.
bc.appendRule( 'U-235',   'Capture', ['Fission', 'U-235FP'] )
bc.appendRule( 'U-236',   ['Capture', 'Np-237'], ['Fission', 'U-235FP'], 'N2N' )
bc.appendRule( 'Np-237', ['Capture', 'Pu-238'], ['Fission', 'U-238FP'] )
bc.appendRule( 'U-238',  ['Capture', 'Np-239'], ['Fission', 'U-238FP'], ['N2N', 'Np-237']  )
bc.appendRule( 'Np-239', ['Fission', 'Pu-239FP'], 'BetaDecay' )
bc.appendRule( 'Pu-238',  'Capture', ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Pu-239',  'Capture', ['Fission', 'Pu-239FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-240',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-241',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'BetaDecay' )
bc.appendRule( 'Pu-242', ['Capture', 'Am-243'], ['Fission', 'Pu-241FP'], 'N2N' )
bc.appendRule( 'Am-241', ['Capture', 'Pu-242', 0.1384], ['Capture', 'Am-242m', 0.2000], ¥
            ['Capture', 'Cm-242', 0.6616], ['Fission', 'Pu-241FP'], ['N2N', 'Pu-240'], 'AlphaDecay' )
bc.appendRule( 'Am-242m', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', ['Decay', 'Pu-242', 0.173],
            ['Decay', 'Cm-242', 0.827] )
bc.appendRule( 'Am-243', ['Capture', 'Cm-244'], ['Fission', 'Pu-241FP'] )
bc.appendRule( 'Cm-242',  'Capture', ['Fission', 'Pu-241FP'], ['N2N', 'Am-241'], 'AlphaDecay' )


info_man.setBurnupChain(bc)
```

図 4-3　燃焼チェーンの入力例

(4) 感度計算に関する核種別の反応タイプの指定

ここでは、核種毎の感度計算を行う際の、核分裂スペクトルの微小変動量、及び計算対象とする反応タイプを指定する。

前半部分では、感度係数の計算を行う核分裂スペクトルの微小変動量を、リスト形式で入力する。後半部分では、感度係数の計算対象となる核種名、及び反応タイプを入力する。入力形式は燃焼チェーンと同様に第1引数に核種名を、第2引数以降に反応タイプを入力する。

この入力はデフォルトから特に変更する必要は無いと思われる。

```
### Sensitivity Calculation Option
so = SensitivityOption()

delta_xi = [-1.00, -0.90, -0.80, -0.70, -0.60, -0.50, -0.40, -0.30, -0.20, -0.10, -0.05,
             0.05,  0.10,  0.20,  0.30,  0.40,  0.50,  0.60,  0.70,  0.80,  0.90,  1.00]
so.setDeltaFissionSpectrum( delta_xi )

# Capture, Nu, Transport, Fission, TotalScattering, ElasticScattering, InelasticScattering
# Mu, N2N, InelasticCrossSection N
so.appendTarget( 'U-235',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-236',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-237',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )
so.appendTarget( 'U-238',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering',
                 'Mu', 'N2N' )

info_man.setSensitivityOption(so)
```

図 4-4　感度計算の対象となる核分裂スペクトルの変動量、核種毎の反応タイプの入力例

4.3.3 core_data.py

このファイルには、炉心に関するデータが含まれる。含まれるデータは

- マテリアル情報
- 領域情報
- 炉心体系
- 炉心構成（マテリアル配置）
- 境界条件

である。以下ではマテリアル情報から順に、入力形式を説明していく。

(1) マテリアル情報

ここでは、計算体系中の領域に割り当てるマテリアルデータセットを定義する。マテリアルの定義は、マテリアルを構成する核種と数密度の組をディクショナリ形式で定義する。

｛核種名：数密度，・・・（必要数だけ）｝

これをマテリアルの数だけ作成する。

```
### Material Set
mat_data = [{ 'Ni-nat.':7.79823e-05, 'Pu-240':0.000598514, 'Mo-nat.':5.96308e-05,
              'U-238':0.00653346,    'Nd-143':0.000275794, 'Pu-242':7.21147e-05,
              'Pu-241':7.98413e-05,  'Na-23':0.00732332,   'Am-241':3.71706e-05,
              'Cr-nat.':0.0024206,   'U-235':1.99109e-05,  'W-nat.':0.000124471,
              'Mn-55':0.000124962,   'Cm-244':1.83577e-05, 'Pu-239':0.00101294,
              'Am-243':1.84334e-05,  'O-16':0.0169491,     'Pu-238':2.06825e-05,
              'Np-237':9.4505e-06,   'Fe-nat.':0.0175173 },                        # IC01, IC02

            { 'Ni-nat.':7.79823e-05, 'Pu-240':0.000666396, 'Mo-nat.':5.96308e-05,
              'U-238':0.00632977,    'Nd-143':0.000275794, 'Pu-242':8.02937e-05,
              'Pu-241':8.88966e-05,  'Na-23':0.00732332,   'Am-241':4.13863e-05,
              'Cr-nat.':0.0024206,   'U-235':1.92901e-05,  'W-nat.':0.000124471,
              'Mn-55':0.000124962,   'Cm-244':2.04398e-05, 'Pu-239':0.00112782,
              'Am-243':2.05241e-05,  'O-16':0.0169639,     'Pu-238':2.30282e-05,
              'Np-237':1.05223e-05,  'Fe-nat.':0.0175173 },                        # OT01, OT02
              .
              .
              .
          ]
material_set = [ Material(x) for x in mat_data ]
#print material_set
```

これが一つのマテリアルを表す

マテリアルの数だけ入力

マテリアルの作成（変更不要）

図 4-5　マテリアルデータセットの入力例

(2) 領域情報

ここでは、計算体系中の領域の情報を作成する。領域情報に必要なデータとして、以下のものが挙げられる。

1. 領域 ID
2. 領域名
3. マテリアル
4. 各サブゾーンに含まれる集合体数
5. 燃料交換オプション（YES/NO：1.0/0.0）

マテリアルには先ほど作成したマテリアルのデータを入力する。

```
### Region Set
region_data = [    [  1, "ICO1", material_set[0],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  2, "ICO2", material_set[0],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  3, "OTO1", material_set[1],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  4, "OTO2", material_set[1],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  5, "ABLU", material_set[2],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  6, "ABLL", material_set[2],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  7, "RBLU", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  8, "RBLM", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [  9, "RBLL", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                   [ 10, "NAFO", material_set[4],  [ 1.0 ], 0.0],
                   [ 11, "GPL1", material_set[5],  [ 1.0 ], 0.0],
                   [ 12, "GPL2", material_set[6],  [ 1.0 ], 0.0],
                   [ 13, "AXSU", material_set[7],  [ 1.0 ], 0.0],
                   [ 14, "AXSL", material_set[8],  [ 1.0 ], 0.0],
                   [ 15, "RDS1", material_set[9],  [ 1.0 ], 0.0],
                   [ 16, "RDS2", material_set[10], [ 1.0 ], 0.0]
                   ]

region_set = [ Region( no=x[0], name=x[1], material=x[2], assembly=x[3], exchange=x[4] ) ¥
              for x in region_data ]
info_man.setRegions(region_set)
#print region_set
```

図 4-6　領域情報の入力例

マテリアル情報と領域情報は、必ずしも上記のような設定を行う必要はない。PSAGEP の入力は、Python スクリプトそのものであり、ユーザが分かりやすい形式で自由に行うことができる。例えば、次のようにマテリアル・オブジェクトと領域オブジェクトを明示的に作成し、InfoManager に追加登録することも可能である。

```
### Material Set
mat1 = Material( { 'Ni-nat.':7.79823e-05, 'Pu-240':0.000598514, 'Mo-nat.':5.96308e-05,
                   'U-238':0.00653346,    'Nd-143':0.000275794, 'Pu-242':7.21147e-05,
                   'Pu-241':7.98413e-05,  'Na-23':0.00732332,   'Am-241':3.71706e-05,
                   'Cr-nat.':0.0024206,   'U-235':1.99109e-05,  'W-nat.':0.000124471,
                   'Mn-55':0.000124962,   'Cm-244':1.83577e-05, 'Pu-239':0.00101294,
                   'Am-243':1.84334e-05,  'O-16':0.0169491,     'Pu-238':2.06825e-05,
                   'Np-237':9.4505e-06,   'Fe-nat.':0.0175173 } )
reg1 = Region( no=1, name="IC01", material = mat1, assembly = [ 1.0, 1.0, 1.0, 1.0 ], exchange = 1)
info_man.appendRegions(reg1)

# 以下同様
```

図 4-7　マテリアル情報と領域情報の指定に関する別の例

(3)　炉心体系

ここでは、CITATION、SAGEP 等で用いられる、メッシュ内の領域数、メッシュ幅を入力する。前半部分では、各メッシュ内の領域数を 2 次元リスト形式で入力する。（メッシュ、領域の関係については CITATION のマニュアルを参照）

```
### Geometry
geom = Geometry()

### Number of division of each mesh
div =    [
         [  2, 3, 3, 3, 2, 4, 4, 3, 2, 5, 4, 4, 2, 3, 2, 2, 4, 4, 4, 5, 3 ], # X AXIS
         [  5, 6, 4, 3, 3, 3, 4, 3, 3, 3, 4, 16, 5],                         # Y AXIS
         [  1 ]                                                              # Z AXIS
         ]

geom.setMeshDivision( div )
```

図 4-8　メッシュ内の領域数の入力例

次に、後半部分では、各メッシュの幅を 2 次元リスト形式で入力する。

```
### Width of each mesh
width =  [
         [ 9.55043, 15.71764, 16.36130, 16.46364,  5.97324, 18.64292, 17.91088, 12.78505,
           4.72722, 21.25165, 17.83599, 18.10268,  4.62130, 13.20161, 10.12392,  3.55860,
          18.97454, 17.49924, 17.37424, 32.60152, 17.22164 ],
         [ 30.0000,  30.0000,  20.0000,  15.0000,  15.0000,  15.0000,  20.0000,  15.0000,
           15.0000,  15.0000,  20.0000,  100.000,  30.0000 ],
         [  1.0000 ]
         ]

geom.setMeshWidth( width )
```

図 4-9　メッシュ幅の入力例

(4) 炉心構成（マテリアル配置）

ここでは、各メッシュに割り当てられるマテリアル ID を、2 次元のリスト形式で入力する。ここで、メッシュに割り当てられる ID は、Region Set の第 1 引数がこれに該当する。

```
### material number of each mesh
material_map = [[ 10,13,13,13,10,13,13,13,10,13,13,13,10,13,13,10,13,13,13,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                [ 14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,16 ]
                ]


geom.setMaterialMap( material_map )
```

図 4-10 炉内のマテリアル配置の入力例

(5) 境界条件

ここでは、CITATION、SAGEP 等で用いられる外部境界の境界条件を入力する。境界条件の入力について、グローバル変数として「REFLECTIVE」、「VACUUM」、「DUMMY」が用意してある。ユーザはこれを用いて入力を行う。（数値による入力も可能だが、数値入力では直感的な理解が難しい為、これらの変数を用いることが望ましい）

入力の方法は 2 通りある。1 つはリスト形式で入力する方法（図 4-10　4、5 行目参照）。もう 1 つは setBoundaryConditionLeft 等のメソッドを用い、方向ごとに入力する方法（下図 6 行目以降）である。

```
#  XYZ:        Left:X-    Top:Y-   Right:X+  Bottom:Y+   Front:Z-    Back:Z+    Black_Region
#  RZ:         Left:X-    Top:Y-   Right:X+  Bottom:Y+  Dummy(0.0)  Dummy(0.0)  Black_Region
#  dir:        BC_LEFT    BC_TOP   BC_RIGHT  BC_BOTTOM   BC_FRONT   BC_BACK     BC_BLACK
#boundary = [ REFLECTIVE,  VACUUM,   VACUUM,   VACUUM,   REFLECTIVE, REFLECTIVE,  DUMMY ]
#geom.setBoundaryCondition( boundary )

geom.setBoundaryConditionLeft( REFLECTIVE )
geom.setBoundaryConditionTop( VACUUM )
geom.setBoundaryConditionRight( VACUUM )
geom.setBoundaryConditionBottom( VACUUM )
geom.setBoundaryConditionFront( REFLECTIVE )
geom.setBoundaryConditionBack( REFLECTIVE )
geom.setBoundaryConditionBlackRegion( DUMMY )
info_man.setGeometry( geom )
```

図 4-11　境界条件の入力例

4.3.4　ユーザーインプット

PSAGEP 実行時の引数として、ユーザが指定するインプットは br_input.py や burnup_input.py 等のファイルであり、解析を行うユーザ、解析シナリオ毎に異なる。このファイルには、次の情報をが含まれている。

- 燃焼ステップ情報
- 収束条件
- シナリオ名
- 断面積ファイル、テスト用リファレンスのパス情報
- 感度計算オプション

以下では燃焼ステップ情報から順に、入力形式を説明していく。

(1) 燃焼ステップ情報

熱出力、冷却期間、運転日数を、燃焼ステップ毎に入力する。ここで入力した数が計算を行う燃焼ステップ数となる。また、各サイクルの情報はここで入力した順番になる。

```
### Burnup Cycles
#                Cooling   Operation Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling =  0*DAYS, operating = 548*DAYS,
                                substep = 20 )  # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                substep = 20 )  # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                substep = 20 )  # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS,
                                substep = 20 )  # CYC-04
```

図 4-12　燃焼ステップの入力例

(2) 収束条件

SAGEP-BURN システム内の計算（CITATION、SAGEP）で用いられる収束条件を入力する。入力値の変更を行う際には、code、type の部分は変更せず、value の入力のみ変更すること。

```
### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )

info_man.setConvergenceCondition(cc)
```

図 4-13　収束条件の入力例

(3)　シナリオ名

解析シナリオ名を指定する。シナリオ名には

- BreedingRatio（増殖比）
- BurnupReactivity（燃焼反応度損失）
- K-Effective（実行増倍率：未対応）
- NumberDensity（原子数密度：未対応）
- ReactionRate（反応率：未対応）
- NaVoidReactivity（ナトリウムボイド反応度価値：未対応）

がある。また、この中で実行増倍率、反応率、及び反応度価値に関する計算は感度計算を行うタイミング（BOC or EOC）の指定を行う必要がある。この場合の入力には下図のコメントアウトされている部分を参照のこと。

```
### Scenario Name
scenario_name = 'BreedingRatio'
#calculation_timing =  'BOC'
info_man.setScenarioName(scenario_name)
#info_man.setCalculationTiming(calculation_timing)
```

図 4-14　解析シナリオ名の入力例

(4) 断面積ファイル、テスト用リファレンスのパス情報

ここでは以下のデータに対するパスの指定を行う。

- cross_section_file_path：実効ミクロ断面積が存在するパス
- data_base_path：計算初期データベースが存在するパス
- reference_path：テスト用リファレンスが存在するパス

```
### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/br/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/br/Burn/'

info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )
```

図 4-15　パス指定の入力例

(5) 感度計算オプション

ここでは、感度計算をする際に必要となる入力を与える。ここでは一例として増殖比を計算する祭に必要となる親核種、娘核種、その他の核種の指定方法を下図に示す。現在は増殖比計算のみに対応している。（燃焼反応度損失の計算では、感度計算に特有の入力は存在しない）

```
### Calculation Breeding Ratio Option
so = info_man.getSensitivityOption()

fertile = ['U-238', 'Pu-240', 'Pu-238']
daughter = ['Pu-239', 'Pu-241']
other = ['U-235', 'U-236', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Np-237', 'Np-239', 'Cm-242']
so.setBreedingOption(fertile, daughter, other)

info_man.setSensitivityOption(so)
```

図 4-16　感度解析オプションの入力例

4.4 解析の実行

前節では、計算に必要な入力値についての説明を行った。本節では解析の実行手順を説明する。

4.4.1 解析の実行の仕方について

まず、実行用のディレクトリを作成する。ここでは、ホームディレクトリの直下に psagep-works というディレクトリを作るものとする。

```
# mkdir ~/psagep-works
# cd ~/psagep-works
```

次に、インプットを作成する。サンプルインプットが psagep/data 以下にあるので、これをコピーして用いる。

```
# cp /usr/local/psagep/work/psagep/data/br-input.py ./br-input.py
```

では、psagep コマンドを実行してみる。まず引数無しで実行する。すると、コマンドの簡単な説明が見られる。

```
# psagep.py
usage: psagep.py [options] <input>
    options:
        --debug, -d     debug mode
        --all,          -A   All calculation is executed(default)
        --forward, -f   Only Forward Calcuration is executed
        --adjoint, -a   Only Forward Calcuration is executed

        --edit, -e   Edit mode.(edit only)

    input:     user input file name (eg. br_input, burnup_input)
```

オプションの説明を以下に示す。

表 4-1　psagep オプションの説明

オプション	説明
--debug -d	デバッグオプション。インプットで指定したリファレンスディレクトリにある計算結果と、PSAGEP の各ステップにおける計算結果を比較し、機能の確認を行う。
--all -A	Forward、Adjoint の全計算を行う。
--forward -f	Forward 計算のみ行う。
--adjoint -a	Adjoint 計算のみ行う。計算時には Forward 計算で作成されたリスタートファイルと DataBase ファイルが必要。条件を変えて Adjoint 計算のみやり直したい場合に用いる。
--edit -e	計算結果のエディットのみ行う。Adjoint 計算で作成されたリスタートファイルが必要。このオプションが選択された際には、他のオプションは無視され、計算は行われない。

次に、引数を与えて実行する。まず、サンプルインプットを変更せずに用いてデバッグモードで実行し、PSAGEP システムが正常に動作するかどうかを確認する。

計算が正常に終了すると、計算を実行したディレクトリ（ユーザーインプットのあるディレクトリ）に計算結果ができる。

```
# psagep.py -d -A   br_input
# ls
Adjoint_BreedingRatio.dat    DataBase        DataBase-Adjoint-BreedingRatio
DataBase-Forward-BreedingRatio                Forward_BreedingRatio.dat
SNS_BreedingRatio   br_input.py   br_input.pyc
```

SNS_BreedingRatio とリファレンスディレクトリの SNS が完全に一致していれば、PSAGEP は正常に動作している。

4.5 解析結果の評価

解析が終了すると、実行ディレクトリ内に、PSAGEP の解析結果である SNS_{解析ケース} ファイルができる。このファイルには核種、反応タイプ、エネルギー群ごとの感度係数が含まれている。下図に例として SNS ファイルの一部を示す。

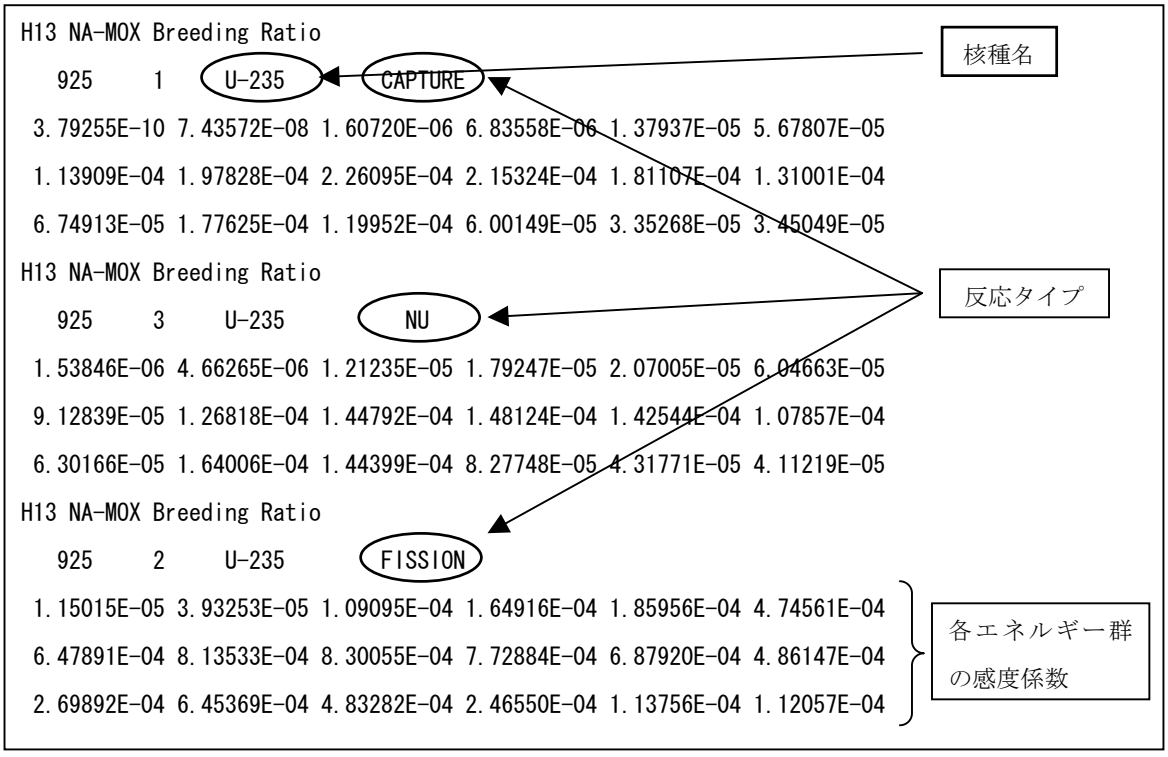


図 4-17　感度計算結果の例

5. おわりに

高速炉の実用化に向けて、燃焼感度解析が果たす役割はきわめて重要であり、同解析を行うための解析コード SAGEP-BURN の有効性が確認されてきた。しかしながら、この燃焼感度の解析は非常に複雑であり、SAGEP-BURN を用いた解析では理論に合わせてユーザが計算ステップをひとつずつ追いかける必要があるため、ユーザへの負担が大きく解析作業が極めて非効率的という問題があった。また、解析対象によって計算ステップが変わることや、物理的意味を分析する際には計算ステップを分解する必要があることから、現在の燃焼感度解析コードをより使いやすくシステム化する必要があった。このため、本研究では、オブジェクト指向とスクリプト言語の技術を利用して、燃焼感度解析コードのシステム化作業を実施した。

平成 15 年度に実施したフェーズ 1 開発では、既存の SAGEP-BURN システムのどこに問題があるのかをプロセス分析を通じて明らかにし、新システムではどのような形態が望ましいかを検討した。その結果、オブジェクト指向スクリプト言語 Python を用い、既存システムのカプセル化部とそれらの制御部からなる二階層制御モデルの設計と、カプセル化層の実装を行った。本実装で、既存の SAGEP-BURN システムの全機能は Python 言語レベルで制御可能となり、特に入出力に関わる部分の柔軟性が大幅に向上した。なお、システムモジュールを含め、計算ロジックには変更は施しておらず、今回のシステムを用いて従来の解析結果と同一の計算結果が得られることを確認した。

平成 16 年度のフェーズ 2 開発では、制御層の詳細設計と実装を行い、PSAGEP(Python-wrapped SAGEP-burn)システムとして完成した。PSAGEP では、解析のために必要なバックエンド計算コードが組み込まれており、ユーザはその存在を意識しなくても燃焼感度解析が実施できる。今回の開発では、増殖比と燃焼反応度損失の二種類の解析シナリオを実装したが、異なる解析シナリオについても容易に実装することが可能である。

なお、ここで整備した Python クラスライブラリは、シナリオクラス等の PSAGEP に強く依存した一部のクラスを除けば、PSAGEP とは独立しており、従来コードの再構成プラットフォームとして活用することが可能である。　本システム化整備を実施することで、カプセル化に基づくシステム整備の手法が既存の解析コードを新しいシステム構成に移行する方法として効率的かつ有効であることが確認できた。また、本手法は既存の解析コードをカプセル化するため既存コードの内部構造が残るが、一旦カプセル化してユーザの利用範囲とテストプルグラムを作りこんで固定しておけば、次の段階で内部構造を徐々に新システムに適合するように変更していくことが可能である。

6. 参考文献

1. 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」, JNC TJ9124 93-009 (1993).
2. 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)」, JNC TJ9124 94-007 (1994).
3. 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)
4. http://www.python.org
5. 横山 賢治, 細貝 広視 他, 「高速炉用オブジェクト統合型解析システムの研究開発（１）」日本原子力学会「2003 年秋の大会」 E64 (2003).
6. 横山 賢治, 細貝 広視 他, 「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) 課題及び要素技術の調査」, JNC TN9420 2002-004 (2002).
7. 横山 賢治, 細貝 広視 他, 「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).
8. 日本 XP ユーザグループ,「eXtereme Programming テスト技法　xUnit ではじめる実践 XP プログラミング」, 翔泳社 (2001).
9. 竹政 昭利, 「はじめて学ぶ UML」, ナツメ社 (2003).
10. ScientificPython, http://starship.python.net/~hinsen/ScientificPython/

付録 1　PSAGEP システムの入力データ例

図 A1-1 base.py

```
from sagep.config import *
from sagep.Util.Nuclide import *
from sagep.Util.Material  import *
from sagep.Util.Region import *
from sagep.Util.Geometry import *
from sagep.Util.SensitivityOption import *
from sagep.Util.ConvergenceCondition import *
from sagep.Manager.InfoManager  import *


### InfoManager
info_man = InfoManager()
```

図 A1-2 br-input.py

```
from data.core_data import *


### Burnup Cycles
#          Thermal-Power    Cooling   Operation   Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling =  0*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-04


### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )

info_man.setConvergenceCondition(cc)


### Scenario Name
scenario_name = 'BreedingRatio'

info_man.setScenarioName(scenario_name)


### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/br/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/br/Burn/'

info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )


### Calculation Breeding Ratio Option
so = info_man.getSensitivityOption()

fertile = ['U-238', 'Pu-240', 'Pu-238']
daughter = ['Pu-239', 'Pu-241']
other = ['U-235', 'U-236', 'Pu-242', 'Am-241', 'Am-242m', 'Am-243', 'Np-237', 'Np-239', 'Cm-242']
so.setBreedingOption(fertile, daughter, other)

info_man.setSensitivityOption(so)


print "InfoManager is built for a default input set.¥n"
```

図 A1-3 burnup-input.py

```
from data.core_data import *


### Burnup Cycles
#          Thermal-Power    Cooling   Operation   Sub-Step
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling =  0*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-01
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-02
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-03
info_man.appendBurnupCycleData( thermalpower = 3570.00E+06, cooling = 42*DAYS, operating = 548*DAYS, substep = 20 )  # CYC-04


### Convergence Condition
cc = ConvergenceCondition()
cc.appendRule( code = 'Citation', type = 'MaxOuterIterations', value = 300 )
cc.appendRule( code = 'Citation', type = 'InnerCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Citation', type = 'OuterCriterion', value = 1.0E-05 )
cc.appendRule( code = 'Sagep', type = 'MaxInnerIterations', value = 1000 )
cc.appendRule( code = 'Sagep', type = 'MaxOuterIterations', value = 200 )
cc.appendRule( code = 'Sagep', type = 'MaxCalculationTime', value = 2000 )
cc.appendRule( code = 'Sagep', type = 'InnerCriterion', value = 1.0E-04 )
cc.appendRule( code = 'Sagep', type = 'CutOffCondition', value = 5.0E-02 )

info_man.setConvergenceCondition(cc)


### Scenario Name
scenario_name = 'BurnupReactivity'

info_man.setScenarioName(scenario_name)


### Path Infomation
cross_section_file_path = '/project/psagep/sagepburn/burn-up/WORK/'
data_base_path = '/project/psagep/work/hyd/psagep/data/'
reference_path = '/project/psagep/sagepburn/burn-up/Burn/'

info_man.setPath( target = 'cross_section', path = cross_section_file_path )
info_man.setPath( target = 'database', path = data_base_path )
info_man.setPath( target = 'reference', path = reference_path )


print "InfoManager is built for a default input set.¥n"
```

図 A1-4 core_data.py

```
from nuclide_data import *

### Material Set
mat_data = [{ 'Ni-nat.':7.79823e-05, 'Pu-240':0.000598514, 'Mo-nat.':5.96308e-05,
              'U-238':0.00653346,    'Nd-143':0.000275794, 'Pu-242':7.21147e-05,
              'Pu-241':7.98413e-05,  'Na-23':0.00732332,   'Am-241':3.71706e-05,
              'Cr-nat.':0.0024206,   'U-235':1.99109e-05,  'W-nat.':0.000124471,
              'Mn-55':0.000124962,   'Cm-244':1.83577e-05, 'Pu-239':0.00101294,
              'Am-243':1.84334e-05,  'O-16':0.0169491,     'Pu-238':2.06825e-05,
              'Np-237':9.4505e-06,   'Fe-nat.':0.0175173 },                          # IC01, IC02

            { 'Ni-nat.':7.79823e-05, 'Pu-240':0.000666396, 'Mo-nat.':5.96308e-05,
              'U-238':0.00632977,    'Nd-143':0.000275794, 'Pu-242':8.02937e-05,
              'Pu-241':8.88966e-05,  'Na-23':0.00732332,   'Am-241':4.13863e-05,
              'Cr-nat.':0.0024206,   'U-235':1.92901e-05,  'W-nat.':0.000124471,
              'Mn-55':0.000124962,   'Cm-244':2.04398e-05, 'Pu-239':0.00112782,
              'Am-243':2.05241e-05,  'O-16':0.0169639,     'Pu-238':2.30282e-05,
              'Np-237':1.05223e-05,  'Fe-nat.':0.0175173 },                          # OT01, OT02

            { 'Ni-nat.':7.79823e-05, 'Mo-nat.':5.96308e-05, 'Na-23':0.00732332,
              'U-235':2.4628e-05,    'U-238':0.00808131,    'W-nat.':0.000124471,
              'Fe-nat.':0.0175173,   'O-16':0.0162119,      'Cr-nat.':0.0024206,
              'Mn-55':0.000124962 },                                                 # ABLU, ABLL

            { 'Ni-nat.':6.4778e-05,  'Mo-nat.':4.95338e-05, 'Na-23':0.00661882,
              'U-235':3.3389e-05,    'U-238':0.0109561,     'W-nat.':0.000103395,
              'Fe-nat.':0.0145512,   'O-16':0.021979,       'Cr-nat.':0.00201073,
              'Mn-55':0.000103803 },                                                 # RBLU, RBLM, RBLL

            { 'Ni-nat.':0.00143655,  'Mo-nat.':0.00010992,  'Na-23':0.0203057,
              'Fe-nat.':0.00458722,  'Cr-nat.':0.00121654,  'Mn-55':0.000134329 },   # NAF0

            { 'Ni-nat.':0.00376282,  'Mo-nat.':0.000287732, 'Na-23':0.00722756,
              'Fe-nat.':0.0120114,   'Cr-nat.':0.00318543,  'Mn-55':0.000351733 },   # GPL1

            { 'Ni-nat.':0.0029681,   'Mo-nat.':0.000226961, 'Na-23':0.00685516,
              'Fe-nat.':0.00947455,  'Cr-nat.':0.00251265,  'Mn-55':0.000277445 },   # GPL2

            { 'Ni-nat.':0.00269258,  'Mo-nat.':0.000206026, 'Na-23':0.00722164,
              'B-11':0.0421443,      'B-10':0.0105361,      'C-12':0.0131701,
              'Cr-nat.':0.0022802,   'Mn-55':0.000251777,   'Fe-nat.':0.008598 },    # AXSU

            { 'Ni-nat.':0.0123677,   'Mo-nat.':0.000946329, 'Na-23':0.00555674,
              'Fe-nat.':0.0394927,   'Cr-nat.':0.0104735,   'Mn-55':0.00115647 },    # AXSL

            { 'Ni-nat.':0.0117553,   'Mo-nat.':0.000899472, 'Na-23':0.006383,
              'Fe-nat.':0.0375373,   'Cr-nat.':0.00995491,  'Mn-55':0.00109921 },    # RDS1

            { 'Ni-nat.':0.00342018,  'Mo-nat.':0.000261699, 'Na-23':0.00428467,
              'B-11':0.0488005,      'B-10':0.0122001,      'C-12':0.0152502,
              'Cr-nat.':0.00289636,  'Mn-55':0.000319813,   'Fe-nat.':0.0109214 }]   # RDS2


### Region Set
material_set = [ Material(x) for x in mat_data ]
#print material_set

region_data = [      [  1, "IC01", material_set[0],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  2, "IC02", material_set[0],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  3, "OT01", material_set[1],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  4, "OT02", material_set[1],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  5, "ABLU", material_set[2],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  6, "ABLL", material_set[2],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  7, "RBLU", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  8, "RBLM", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [  9, "RBLL", material_set[3],  [ 1.0, 1.0, 1.0, 1.0 ], 1.0],
                     [ 10, "NAF0", material_set[4],  [ 1.0 ], 0.0],
                     [ 11, "GPL1", material_set[5],  [ 1.0 ], 0.0],
                     [ 12, "GPL2", material_set[6],  [ 1.0 ], 0.0],
                     [ 13, "AXSU", material_set[7],  [ 1.0 ], 0.0],
                     [ 14, "AXSL", material_set[8],  [ 1.0 ], 0.0],
                     [ 15, "RDS1", material_set[9],  [ 1.0 ], 0.0],
                     [ 16, "RDS2", material_set[10], [ 1.0 ], 0.0]
                    ]

region_set = [ Region( no=x[0], name=x[1], material=x[2], assembly=x[3], exchange=x[4] ) for x in region_data ]
info_man.setRegions(region_set)
#print region_set
```

```
### Geometry
geom = Geometry()

### Number of division of each mesh
div = [
          [  2, 3, 3, 3, 2, 4, 4, 3, 2, 5, 4, 4, 2, 3, 2, 2, 4, 4, 4, 5, 3 ], # X AXIS
          [  5, 6, 4, 3, 3, 3, 4, 3, 3, 3, 4, 16, 5],                          # Y AXIS
          [  1 ]                                                               # Z AXIS
          ]

geom.setMeshDivision( div )

### Width of each mesh
width = [
          [ 9.55043, 15.71764, 16.36130, 16.46364,  5.97324, 18.64292, 17.91088, 12.78505,
            4.72722, 21.25165, 17.83599, 18.10268,  4.62130, 13.20161, 10.12392,  3.55860,
           18.97454, 17.49924, 17.37424, 32.60152, 17.22164 ],
          [ 30.0000,  30.0000,  20.0000,  15.0000,  15.0000,  15.0000,  20.0000,  15.0000,
            15.0000,  15.0000,  20.0000,  100.000,  30.0000 ],
          [  1.0000 ]
          ]

geom.setMeshWidth( width )

### material number of each mesh
material_map = [[ 10,13,13,13,10,13,13,13,10,13,13,13,10,13,13,10,13,13,13,15,16 ],
                  [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                  [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                  [ 10, 5, 5, 5,10, 5, 5, 5,10, 5, 5, 5,10, 5, 5,10, 5, 5, 7,15,16 ],
                  [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                  [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                  [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                  [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                  [ 10, 1, 1, 1,10, 1, 1, 1,10, 2, 2, 2,10, 3, 3,10, 4, 4, 8,15,16 ],
                  [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                  [ 10, 6, 6, 6,10, 6, 6, 6,10, 6, 6, 6,10, 6, 6,10, 6, 6, 9,15,16 ],
                  [ 10,11,11,11,10,11,11,11,10,11,11,11,10,11,11,10,11,11,12,15,16 ],
                  [ 14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,16 ]
                  ]

geom.setMaterialMap( material_map )


### boundary condition
#  XYZ:       Left:X-    Top:Y-   Right:X+  Bottom:Y+   Front:Z-    Back:Z+    Black_Region
#  RZ:        Left:X-    Top:Y-   Right:X+  Bottom:Y+  Dummy(0.0)  Dummy(0.0)  Black_Region
#  dir:       BC_LEFT    BC_TOP   BC_RIGHT  BC_BOTTOM   BC_FRONT   BC_BACK     BC_BLACK
#boundary = [ REFLECTIVE,  VACUUM,   VACUUM,   VACUUM,   REFLECTIVE, REFLECTIVE,  DUMMY ]
#geom.setBoundaryCondition( boundary )

geom.setBoundaryConditionLeft( REFLECTIVE )
geom.setBoundaryConditionTop( VACUUM )
geom.setBoundaryConditionRight( VACUUM )
geom.setBoundaryConditionBottom( VACUUM )
geom.setBoundaryConditionFront( REFLECTIVE )
geom.setBoundaryConditionBack( REFLECTIVE )
geom.setBoundaryConditionBlackRegion( DUMMY )
info_man.setGeometry( geom )
```

図 A1-5 nuclide_data.py

```
from base import *


### Energy group Info.
jfs18g =   [  1.00000000e+07,    6.06530700e+06,    3.67879400e+06,    2.23130200e+06,
              1.35335300e+06,    8.20850000e+05,    3.87742094e+05,    1.83156406e+05,
              8.65169531e+04,    4.08677109e+04,    1.93045391e+04,    9.11882031e+03,
              4.30742480e+03,    2.03468396e+03,    9.61116516e+02,    4.53999298e+02,
              2.14454102e+02,    1.01300903e+02,    9.99999975e-06]
info_man.setEnergyBoundary( jfs18g )


### Nuclide Set
nuclide_names = [ 'U-235',  'U-236',  'Np-237',  'U-238',   'Np-239',  'Pu-238',   'Pu-239',
                  'Pu-240', 'Pu-241', 'Pu-242',  'Am-241',  'Am-242m', 'Am-243',   'Cm-242',
                  'Cm-243', 'Cm-244', 'Cm-245',  'U-235FP', 'U-238FP', 'Pu-239FP', 'Pu-241FP',
                  'O-16',   'Na-23',  'Cr-nat.', 'Mn-55',   'Fe-nat.', 'Ni-nat.',  'Mo-nat.',
                  'Nd-143', 'W-nat.', 'B-10',    'B-11',    'C-12' ]
nuclide_set = [     Nuclide(name=x) for x in nuclide_names ]


info_man.setNuclides( nuclide_set )

#print nuclide_set

### Burnup Chain
bc = BurnupChain()
#           Fertile Name, Reaction Type, [ Reaction Type, Daughter], [ Reaction Type, Daughter, Fraction ]
#           If Reactin Type is Fission, you need to input Daughter.
bc.appendRule( 'U-235',   'Capture', ['Fission', 'U-235FP'] )
bc.appendRule( 'U-236',   ['Capture', 'Np-237'], ['Fission', 'U-235FP'], 'N2N' )
bc.appendRule( 'Np-237', ['Capture', 'Pu-238'], ['Fission', 'U-238FP'] )
bc.appendRule( 'U-238',  ['Capture', 'Np-239'], ['Fission', 'U-238FP'], ['N2N', 'Np-237']  )
bc.appendRule( 'Np-239', ['Fission', 'Pu-239FP'], 'BetaDecay' )
bc.appendRule( 'Pu-238',  'Capture', ['Fission', 'U-238FP'], ['N2N', 'Np-237'] )
bc.appendRule( 'Pu-239',  'Capture', ['Fission', 'Pu-239FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-240',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Pu-241',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'BetaDecay' )
bc.appendRule( 'Pu-242',  ['Capture', 'Am-243'], ['Fission', 'Pu-241FP'], 'N2N' )
bc.appendRule( 'Am-241', ['Capture', 'Pu-242', 0.1384], ['Capture', 'Am-242m', 0.2000], ¥
           ['Capture', 'Cm-242', 0.6616], ['Fission', 'Pu-241FP'], ['N2N', 'Pu-240'], 'AlphaDecay' )
bc.appendRule( 'Am-242m', 'Capture', ['Fission', 'Pu-241FP'], 'N2N', ['Decay', 'Pu-242', 0.173],
           ['Decay', 'Cm-242', 0.827] )
bc.appendRule( 'Am-243', ['Capture', 'Cm-244'], ['Fission', 'Pu-241FP'] )
bc.appendRule( 'Cm-242',  'Capture', ['Fission', 'Pu-241FP'], ['N2N', 'Am-241'], 'AlphaDecay' )
bc.appendRule( 'Cm-243',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Cm-244',  'Capture', ['Fission', 'Pu-241FP'], 'N2N', 'AlphaDecay' )
bc.appendRule( 'Cm-245', ['Fission', 'Pu-241FP'], 'N2N' )

info_man.setBurnupChain(bc)


### Sensitivity Calculation Option
so = SensitivityOption()

delta_xi = [-1.00, -0.90, -0.80, -0.70, -0.60, -0.50, -0.40, -0.30, -0.20, -0.10, -0.05,
             0.05,  0.10,  0.20,  0.30,  0.40,  0.50,  0.60,  0.70,  0.80,  0.90,  1.00]
so.setDeltaFissionSpectrum( delta_xi )

# Capture, Nu, Transport, Fission, TotalScattering, ElasticScattering, InelasticScattering
# Mu, N2N, InelasticCrossSection N
so.appendTarget( 'U-235',     'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-236',     'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-237',     'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-238',     'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-238',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-239',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-240',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-241',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-242',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-241',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-242m',   'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Am-243',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Np-237',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Np-239',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-242',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-243',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cm-244',    'Capture', 'Nu', 'Fission', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
```

```
so.appendTarget( 'O-16',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Na-23',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Cr-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Mn-55',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Fe-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Ni-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Zr-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Mo-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Nd-143',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'W-nat',    'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pb-nat',   'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'B-10',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'B-11',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'C-12',     'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-241FP', 'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-235FP',  'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'U-238FP',  'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )
so.appendTarget( 'Pu-239FP', 'Capture', 'ElasticScattering', 'InelasticScattering', 'Mu', 'N2N' )

info_man.setSensitivityOption(so)
```

付録2 PSAGEP 開発マニュアル

表リスト

図リスト

A2-1. はじめに

本ドキュメントは、開発者向けのドキュメントであり、以下の読者を対象としています。

(1) PSAGEP の解析シナリオの拡充を行う開発者
(2) 既存コードのカプセル化を行い Python によるシステム化を行う開発者

第 2 章では、PSAGEP に対して解析シナリオを追加する方法について説明しています。ここでは、増殖比の燃焼感度計算を対象とした解析シナリオを参考に、燃焼反応度に関する解析シナリオを導出する方法について、具体的に順を追って解説しています。

第 3 章では、既存の計算コードを Python で制御可能とするために、コードのカプセル化やデータファイルのオブジェクト化とその管理方法について説明しています。また、Python によるシステム化の適用例についても紹介しています。

PSAGEP システムについては、情報交換の場として、メーリングリストが開設されています。参加希望の方は、下記までご連絡下さい。

メーリングリスト連絡先：
核燃料サイクル開発機構　大洗工学センター
システム技術開発部　中性子工学グループ
横山賢治　　(e-mail: kyoko@oec.jnc.go.jp)

A2-2. シナリオの追加について

ここでは、解析シナリオの追加方法について解説します。なお本章では、燃焼感度解析及びPython に関して、ある程度の知識があることを前提としています。

A2-2.1 作業の流れ

シナリオの追加は、以下の流れで行います。

まず、シナリオの追加を行うにあたり、対象となるシナリオについて、分析を行う必要があります。このシナリオの分析は、以下の 3 点について行います。

(1) 計算フローの分析
(2) シナリオ固有の入力の分析（各バックエンドの計算の入力について）
(3) シナリオ固有の計算の分析（各バックエンドの計算の計算条件について）

ここで、バックエンドの計算とは、SAGEP-BURN システムで用いられている CITATION や SAGEP 等のコードによる計算を意味します。以下では、SAGEP-BURN システムですでに計算のためのシェルスクリプトが存在している場合について解説していますが、新規に作成する場合にも参考になるでしょう。

上記のシナリオ分析作業は、sagepburn ディレクトリ内の、対象となる感度のディレクトリ（br，burn-up 等）の中のファイルについて行います。これらの違いの調査には、diff –bc や diff –y を用います。以下には、増殖比(以下 br)と燃焼反応度損失(以下 burn-up)の比較を例として説明します。

まず、（1）の計算フローの分析について説明します。Forward 計算の内容は、シナリオによらず変わらないことから、計算フローの分析は Adjoint 計算についてのみ行います。分析手法としては、既に実装されている br の Burn-adj.sh と、新規に追加したいもの（ここでは burn-up）の Burn-adj.sh を比較し、計算フローについて異なる点の抽出を行います。図 A2-1 に br と burn-up の比較結果を示します。この図は Linux のコマンド diff –y の結果です。この図にあるように、計算に過不足がある場合は「<」若しくは「>」であらわされ、ファイル名に変更がある場合は「|」で表されます。この図から、サイクル 4（最終サイクル）及びサイクル 3（最終サイクル－1）で計算の追加があることと、サイクル 3 で実行するシェル名に変更があることが分かります。このように情報をまとめて、計算フローの分析結果とします。

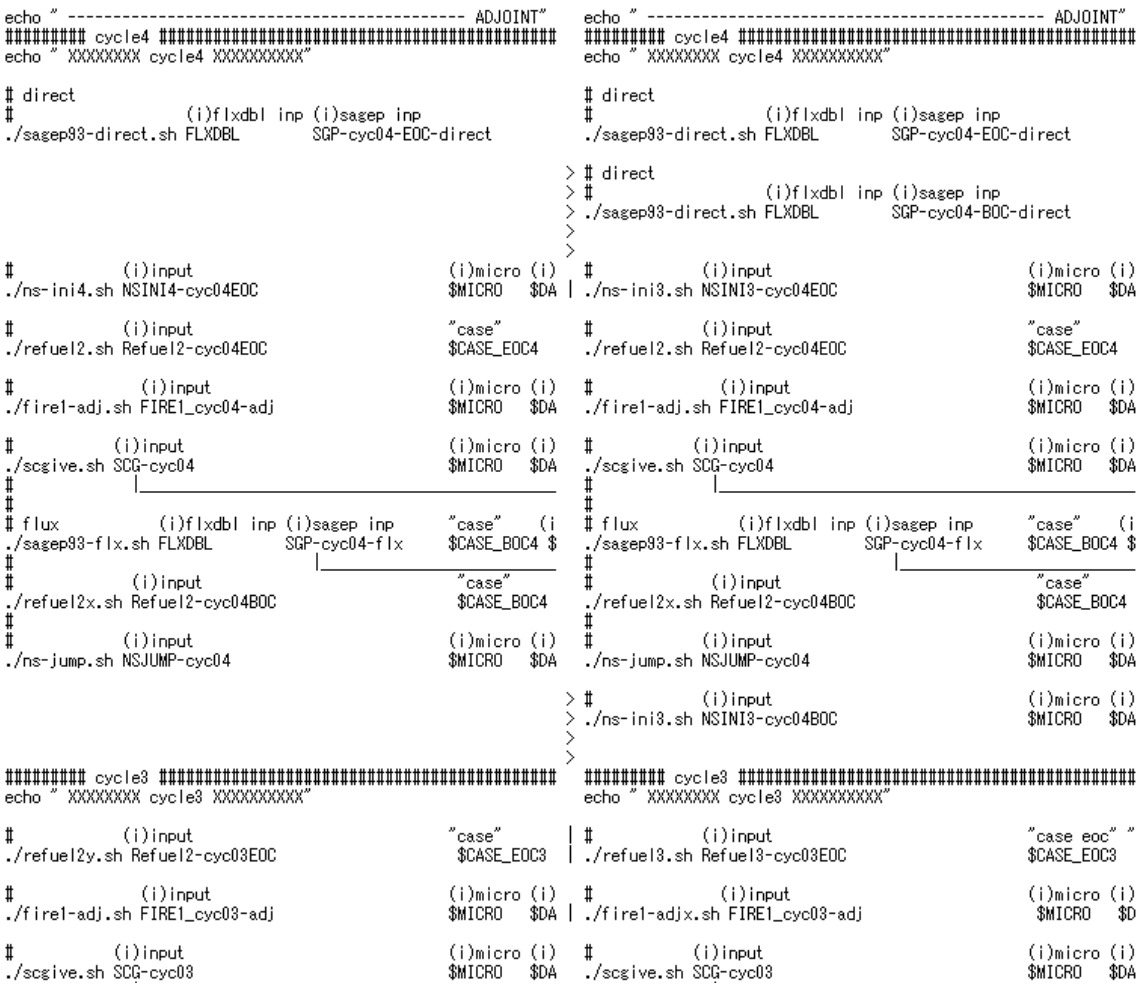


図 A2-1 Burn-adj.sh の変更部分（br と burn-up の比較）

次に、（2）のバックエンドの計算コードの入力ファイルについて、先ほどと同様に、br シナリオと burn-up シナリオについて比較を行います。まず、インプットファイルの過不足を調べます。調査方法は色々ありますが、ここではインプットファイル名を 1 列に並べたファイルを各シナリオについて用意し、先ほどと同様に diff –y を用いて調べています。この結果を図 A2-2 に示します。ここでは、4 サイクル目（最終サイクル）の SAGEP の計算が、EOC だけでなく、BOC でも行われている事と、NSINI4 から NSINI3 に使用コードが変更になっていることが分かります。

```
PDSEDT-DATA                          PDSEDT-DATA
PDSEDT-NUCL                          PDSEDT-NUCL
SGP-cyc01-flx                        SGP-cyc01-flx
SGP-cyc02-flx                        SGP-cyc02-flx
SGP-cyc03-flx                        SGP-cyc03-flx
                                   > SGP-cyc04-BOC-direct
SGP-cyc04-EOC-direct                 SGP-cyc04-EOC-direct
SGP-cyc04-flx                        SGP-cyc04-flx
SAGEP-snslev                         SAGEP-snslev
NSINI4-cyc04EOC                    | NSINI3-cyc04BOC
                                   > NSINI3-cyc04EOC
NSJUMP-cyc02                         NSJUMP-cyc02
NSJUMP-cyc03                         NSJUMP-cyc03
NSJUMP-cyc04                         NSJUMP-cyc04
SCG-cyc01                            SCG-cyc01
SCG-cyc02                            SCG-cyc02
SCG-cyc03                            SCG-cyc03
SCG-cyc04                            SCG-cyc04
```

図 A2-2 バックエンド計算の入力の過不足

続いて、ファイルの内容について調査を行います。方法は色々ありますが、全ての入力ファイルについて diff を行うのが最も簡単な手法になります。この時、以下のように実行すると調べるのが楽になります。

> diff –b br/Burn/(Input File) burn-up/Burn/(Input File) > diff_burn-up/inp/(Input File).diff

ここで、結果をリダイレクトで書き出している点ですが、これには次のようなメリットがあります。ファイルに違いがあった場合、結果がファイルに書き出されますが、違いが無い場合、空のファイルが作成されます。ここで、ls –l を実行すると、ファイルサイズから異なるファイルを探し出すことができます。結果は次のようになります。

```
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc01BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc01EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc02BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc02EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc03BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc04BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:19 Refuel-cyc04EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc01EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc02BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc02EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc03BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc04BOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel2-cyc04EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 Refuel3-cyc03EOC.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SAGEP-snslev.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc01.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc02.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc03.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SCG-cyc04.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc01-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc02-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc03-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc04-BOC-direct.diff
-rw-rw-r--    1 hyd      SE           718  4月 26 16:20 SGP-cyc04-EOC-direct.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 SGP-cyc04-flx.diff
-rw-rw-r--    1 hyd      SE             0  4月 26 16:20 STARTUP.diff
```

図 A2-3　バックエンド計算の入力の変更点について

この結果から、SAGEP の計算の入力のみが異なっていることが分かります。先ほどのファイルの過不足についての結果と、このファイルの内容が、シナリオ固有の入力の分析結果となります。

最後に、（3）の各計算コードの実行シェルについても同様に、接続ファイルに増減、変更が無いか調べます。手順は（2）と同様に行います。結果は次のようになります。

```
cit-adj.sh                      cit-adj.sh
citation.sh                     citation.sh
convmic.sh                      convmic.sh
                              > differ.sh
fire1-adj.sh                    fire1-adj.sh
                              > fire1-adjx.sh
fire1.sh                        fire1.sh
fire2.sh                        fire2.sh
                              > ns-ini3.sh
ns-ini4.sh                      ns-ini4.sh
ns-jump.sh                      ns-jump.sh
pdsedit.sh                      pdsedit.sh
refuel.sh                       refuel.sh
refuel2.sh                      refuel2.sh
refuel2x.sh                     refuel2x.sh
refuel2y.sh                     refuel2y.sh
                              > refuel2z.sh
refuel3.sh                      refuel3.sh
sagep93-direct.sh               sagep93-direct.sh
sagep93-flx.sh                  sagep93-flx.sh
scgive.sh                       scgive.sh
```

図 A2-4 実行シェルの過不足

```
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 cit-adj.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 citation.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 convmic.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 fire1-adj.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 fire1.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 fire2.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 ns-ini4.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 ns-jump.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 pdsedit.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 refuel.sh.diff
-rw-rw-r--   1 hyd      SE          122  4月 26 11:51 refuel2.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 refuel2x.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 refuel2y.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 refuel3.sh.diff
-rw-rw-r--   1 hyd      SE          400  4月 26 11:51 sagep93-direct.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 sagep93-flx.sh.diff
-rw-rw-r--   1 hyd      SE            0  4月 26 11:51 scgive.sh.diff
-rw-rw-r--   1 hyd      SE          104  4月 26 11:51 start-up.sh.diff
```

図 A2-5 実行シェルの変更点について

これらの情報がまとまったら、br のシナリオクラス(BreedingRatioScenario.py)を雛形として、新規シナリオの実装を行います。

A2-2.2 InputGenerator クラスの修正

追加したいシナリオの計算に必要なインプットを作成する部分が、InputGenerator クラスに未実装である場合、これを追加する必要があります。インプットファイルの分析の結果、入力ファイル名の不足や、内容が大きく異なるものがある場合、これに該当する可能性があります。基本的には、scenario_name 変数を用いて、条件分岐させることで対応できます。
InputGenerator クラス中で実際に用いられている部分を、例として次に示します。追加したいシナリオについて、特殊なインプットを作成する場合には、下図のように、シナリオ名で分岐させて対応します。

```
body_format = []
body_value = []

scenario_name = self.info_man.getScenarioName()
so = self.info_man.getSensitivityOption()

# card 8: calculate option
if scenario_name == 'NumberDensity':
    calculation_option = 1
    target_nuclide = so.ndFocusedOption('nuclide')
    jendl_code_number = target_nuclide.id()
    num = 1
    for nuclide in self.info_man.getBurnableNuclides():
        if target_nuclide == nuclide:
            break
        num += 1
    number_of_burnup_chain_nuclide = num
    focused_region = so.ndFocusedOption('region')
else:
    calculation_option = 0
    jendl_code_number = 0
    number_of_burnup_chain_nuclide = 0
    focused_region = 0
```

図 A2-6 InputGenerator クラスの一部（シナリオ名による分岐部）

A2-2.3 ModuleFile クラスの修正

使用する計算コードの定義が存在しない場合、若しくは接続ファイルが異なる場合、ModuleFile クラスの修正を行う必要がある。ModuleFile についての詳細は、3 章のカプセル化を参照してください。

A2-2.4 SensitivityOption クラスの修正

このクラスは、感度計算に固有のデータの格納、提供を行います。現在は共通のデータ（核種毎の計算を行う感度、核分裂スペクトルの摂動量）及び、増殖比に固有のデータについてのメソッドのみ実装されています。

このクラスはコンテナなので、基本的にはデータの格納、提供のみを行います。新規にメソッドを追加する場合も、基本的にはこのルールに従ってください。

A2-2.5 シナリオの追加（Adjoint シナリオの作成）

シナリオの追加作業は、基本的には Burn-adj.sh の内容を Python スクリプトで記述しなおすという作業になります。一から新規にシナリオを作成する場合は、Burn-adj.sh から、まったく新しくシナリオクラスを作成する必要があります。ですが、今回は既に br 及び burn-up シナリオが存在しているので、これを雛形として、追加シナリオを作成します。

シナリオ名には、以下のものが用意してあります。

- K-Effective : 実効増倍率
- BurnupReactivity : 燃焼反応度損失（実装済み）
- NumberDensity : 原子数密度
- BreedingRatio : 増殖比（実装済み）
- ReactionRate : 反応率
- NaVoidReactivity : Na ボイド反応度

これらの名前は InputGenerator、Scenario クラスの内部で用いられています。ここに上げた感度の解析シナリオを新たに作成する場合には、これらの名前を用いてください。
（NumberDensityScenario.py など）

次に、シナリオクラスの基本的な構造について説明します。シナリオクラスは、以下の 4 パートからなります。

- 定義部（計算モジュール、ファイルマネージャなど）
- 制御部（計算実行モードの制御）
- 計算実行部（Forward、Adjoint 計算の実行）
- デバッグ関数

これらの内､Adjoint 計算実行部以外は､各シナリオクラスのスーパークラスである Scenario クラスに含まれており、基本的に変更せずに使うことができます。

計算実行部は、サイクルについてのメインループと、各計算の実行部からできています。ここで、ある特定のサイクル（最終サイクル、第 1 サイクルなど）で特別な計算を行う場合は、そのサイクルについて条件分岐を行います。

個別の計算実行部は、

- 接続するファイル定義の作成
- ファイルの接続
- 計算の実行
- 計算結果の取り出し

の 4 ステップから成り立っています。この例を図 A2- 7 に示します。

シナリオの構築は、各計算コードの実行シェルに対応するように、それぞれの計算コードについて計算実行部を作成し、これを Burn-adj.sh と同じ様に並べるという作業になります。

```
(メインループ：サイクルについてループ)
for cy in range(total_cycle_number,0,-1):
  (接続ファイル定義：InputGenerator から作成)
  sagep_direct_inp = self.ig.generate( code="sagep", cycle=cy, ・・・)
  (接続ファイル定義：FileManager から取り出し)
  flux_forward = self.fm.select( code="citation", type=・・・)
  (ファイル接続)
  self.sagep.input( [sagep_direct_inp, flux_forward, ・・・])
  (計算実行)
  self.sagep.run( casename(cy, "EOC") )
  (計算結果をモジュールクラスから取り出し)
  sagep_direct_outfiles = self.sagep.outfiles()
  (計算結果を FileManager に格納)
  self.fm.add(sagep_direct_outfiles)
  (デバッグ実行：デバッグモード指定時のみ)
  if mode == "Debug"
    self.debugFunction(code='sagep', cycle=cy, ・・・)
```

図 A2-7　Scenario クラス中の個別計算実行部（例：sagep コード実行部）

A2-2.6　テスト実行

シナリオの作成が終わったら、PSAGEP を実行する前に、テストを行います。この手順について説明します。

まず始めに、バックエンドの計算コードが正しく作られるかどうかテストを行います。psagep/data ディレクトリに、追加したシナリオに対応した PSAGEP のユーザーインプットを作成します。次に、psagep/test ディレクトリにある、InputGeneratorTest.py クラスの修正を行います。まず、以下の一文を変更します。ここでは、burnup_input というファイルを作成したと仮定します。

from data.br_input import *　→　from data.burnup_input import *

次に、InputGeneratorTest クラスが、適切に全てのインプットについてテストを行うように、InputGeneratorTest クラスの内部を修正します。図 A2-8 に InputGeneratorTest クラスの一部を示します。

```
def testInputSAGEP(self):
  for cy in [ 1, 2, 3, 4]:
    (InputGenerator で入力作成)
    sagep_inp=self.ig.generate(code="sagep",cycle=cy,case="BOC",option="flux")
    (テスト用リファレンス入力の名前を指定)
    refname=burn_dir[self.scenario_name] + "SGP-cyc%02d-flx" % cy
    print refname
    refname2=burn_dir[self.scenario_name] + "SAGEP-snslev"
    os.system( "cat %s %s > /tmp/sagep.test " % (refname, refname2) )
    (テスト実行)
    assert sagep_inp.has_same_contents_with(RefFile("/tmp/sagep.test"), ¥
                'Breeding'),sagep_inp.diff_from( RefFile( "/tmp/sagep.test") )
```

図 A2-8　InputGeneratorTest クラスのテスト実行部（SAGEP コード）

図に示すように、InputGeneratorTest クラスのテスト実行部は

- InputGenerator クラスによるバックエンドコードのインプット作成
- テストのターゲットとなるリファレンスインプットの指定
- テスト実行

の 3 ステップから成り立っています。これをサイクル数、及び BOC、EOC 等を組み合わせて、新規に追加するシナリオの入力全てを網羅するように修正し、テストを実行します。この際に、サンプルインプットの入力値に、サイクル、使用計算コードによって代わるはずが無いデータが変わっていない事を確認しておいてください。

入力ファイルが正しく作られていることを確認したら、次はシナリオクラスが正しくで動作するかかどうか、psagep コマンドをデバッグモードで実行して確かめます。この時に、事前にサンプル計算を実行して、テスト用のリファレンスアウトプットを作成しておいて下さい。リファレンスアウトプットの作成方法は、sagepburn/burn-up/Burn ディレクトリで、Burn-normal.sh 及び、Burn-adj.sh を実行することで作成できます。また、SAGEP-BURN の最終結果については Burn ディレクトリでの計算が全て終わった後に、sagepburn/burn-up/table ディレクトリで mk_sns.sh を実行することで作成できます。

リファレンスアウトプットの作成が終わったら、psagep コマンドをデバッグモードで実行します。実行方法は次のようになります。

> psagep.py –d (Input File)

デバッグモードでの実行が正常終了したら、シナリオの作成は終了です。

A2-3. 既存コードのカプセル化について

本章では、既存コードを Python によりカプセル化する方法について、チュートリアル形式で説明します。本例題を通じて既存コードのカプセル化を実際におこない、Python 言語で取り扱う方法について学びましょう。

A2-3.1 カプセル化のメリット

カプセル化のメリットを一言で表すと、「抽象化による簡素化」と表現することが出来ます。つまり、詳細な部分について知る必要が無く、面倒な部分をシステム側で自動的に適切に処理する仕組みをつくることができます。例えば、計算コードに対する入力を、何番の I/O ユニットに接続するか、あるいはその計算を行うための前準備をどうするか等、ユーザが詳細について知る必要を無くすることができるようになります。これを活用すると、従来はシェルスクリプト等で複雑に制御されていた計算を Python で置き換えることにより、制御を簡単化したりより柔軟な制御方法を導入したりすることができるでしょう。

以下の例題を通じて、この抽象化によるメリットを是非とも体感して下さい。

A2-3.2 カプセル化の例題

では早速、実際に既存コードのカプセル化を行ってみましょう。ここでは、図 A2-9 に示す Fortran コードを対象とします。

```
program MyFortCode
read(5,*) imax, jmax
do j=1, jmax
   do i=1, imax
      read(1,*)   k
      write(6,*) i, j, k
      write(10,*) i, j, k
   enddo
enddo
end
```

図 A2-9　カプセル化対象の MyFortCode.f

このコードの入出力は、表 A2-1 に示すものであると仮定します。なお、表 A2-1 において括弧内のものは、システム側で自動設定されることを示しています。

表 A2-1　MyFortCode に関するファイル入出力

ユニット番号	入出力	作成コード	ファイル内容（タイプ）
1	入力	CODE1	COUNT_DATA
5	入力	—	(USER_CARD)
6	出力	—	(EDIT_LIST)
10	出力	(MYFORTCODE)	(COUNT_SUMMARY_DATA)

図 A2-9 で示される MyFortCode のコンパイル済みの実行モジュールが、/users/test/MyFortCode であると仮定します。この場合、MyFortCode のカプセル化クラスは図 A2-10 のようになります。これらの定義が書かれたファイル MyModuleFile.py は、ModuleFile.py が保存されているディレクトリ ($prefix/sagep/File/) に存在しているとします。（実際に作業を行う場合は、上記の場所に保存してください。）

では、Python インタープリタを対話的に用いて、MyFortCode を実行してみましょう。

まず、Python インタープリタを起動します。

```
% python
Python 2.3.3 (#1, Apr 14 2004, 20:41:04)
[GCC 3.2.2 20030222 (Red Hat Linux 3.2.2-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

次に、先ほどの MyModuleFile.py の定義部分を読み込みます。

```
>>> from sagep.File.MyModuleFile import *
>>>
```

問題がなければ上記のようにプロンプトが表示されます。ここでエラーメッセージが出る場合は、MyModuleFile.py に記述ミスがあると思われるので再度チェックして下さい。

```
 1 : from sagep.File.ExecutableFile import *
 2 : from sagep.File.CardFile import *
 3 : from os.path import *
 4 : import string
 5 :
 6 : class MyFortCode(ExecutableFile):
 7 :       "Encapsulating class for MyFortCode"
 8 :       def __init__(self, case=""):
 9 :             ExecutableFile.__init__(self, "/users/test/MyFortCode", case )
10 :
11 :             # input file(s)
12 :             self.defineio(unit=1, mode="i", generated="CODE1", type="COUNT_DATA", comment="Counting Data")
13 :
14 :             # output file(s)
15 :             self.defineio(unit=10, mode="o", type = "COUNT_SUMMARY_DATA", comment="Counting Summary Data")
16 :
17 :             # user card and edit list
18 :             self.defineio(unit=5, mode="i", type = "USER_CARD", comment="User Input Cards")
19 :             self.defineio(unit=6, mode="o", type = "EDIT_LIST", comment="Editing List")
```

図 A2-10　カプセル化クラス　MyFortCode (ファイル名を MyModuleFile.py として下さい)

※　行番号は、便宜上付けたもので実際のファイルには含まれておりません。

次に、カプセル化したオブジェクトを実体化します。実体化したオブジェクトのことをインスタンスと呼びます。下の例では、code と名前で計算コードオブジェクトのインスタンスを作成しています。

```
>>> code = MyFortCode()
>>>
```

次に、計算コードに与える入力ユーザカード(5 番ファイル)とデータファイル(1 番ファイル)を準備しましょう。個々では、入力ユーザカードは CardFile クラスを用いて動的に作成してみます。

```
>>> inp_text = [ "2  3" ]
>>> inp = CardFile( inp_text )
>>> inp.show()
===== CardFile  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
file <type 'file'> = <closed file '/tmp/psagep.test.22605/sagepCjmiEF',
mode 'w' at 0x400755e0>
fixed_path_flag <type 'bool'> = False
full_path_name = /tmp/psagep.test.22605/sagepCjmiEF
gen_code =
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code =
type = USER_CARD
===== CardFile  (END) =======
>>>
```

CardFile オブジェクトを作成する場合は、コンストラクタ起動時（上から 2 行目）の引数に、入力カード内容の文字列配列を指定します。3 行目では、show()メソッドを起動することにより、オブジェクト内部の全ての変数を表示して、内容を確認しています。今回の場合では、テンポラリ領域 (/tmp ディレクトリ以下)において、自動的に一時ファイルが作成されていることが分かります。

次に、1 番ファイルを準備しましょう。今度は、すでにあるファイルを対象とします。/users/test/fort1.dat　というファイルがあると仮定します。その内容は図 A2-11 の通りであるとします。

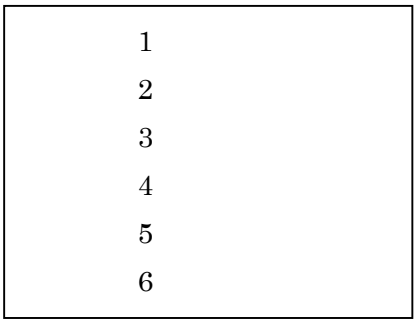


図 A2-11　　fort1.dat の内容

それでは、MyFortCode.inp を抽象データ化するために、DataFile オブジェクトでカプセル化してみましょう。

```
>>> dat = DataFile("/users/test/fort1.dat")
>>> dat.show()
===== DataFile  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/MyFortCode.inp
gen_code =
is_temporary <type 'bool'> = False
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target_code =
type =
===== DataFile  (END) =======
>>>
```

さて、入力ファイルについてはほぼ準備が終わりましたが、実は dat オブジェクトに関してもう少し準備が必要です。

図 A2-10 の 12 行目で定義しているとおり、MyFortCode の 1 番ユニットに接続されるファイルは、計算モジュール”CODE1”で作成された” COUNT_DATA”というファイルタイプ属性を持つ「データファイル（オブジェクト）」を要求しています。言い換えると、複数のデータ

ファイル（オブジェクト）を用意した場合、そのファイルを作成した計算コードと、そのファイルタイプ属性が完全に一致するものが自動的に選択されるという仕組みになっています。したがって、ここでは明示的に dat オブジェクトに作成計算コードとファイルタイプの属性を設定しておきます。

```
>>> dat.set_generate_code("CODE1")
>>> dat.set_filetype("COUNT_DATA")
>>> dat.show()
===== DataFile  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
contents =
data =
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/fort1.dat
gen_code = CODE1
is_temporary <type 'bool'> = False
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target_code =
type = COUNT_DATA
===== DataFile  (END) =======
>>>
```

上記の例では、dat オブジェクトの内部変数である gen_code と type がそれぞれ設定されていることが分かります。（ただし、これらは内部変数なので、ユーザ側から直接操作するのではなく、必ず set_generate_data()等のメソッドを使うようにして下さい。）

では、いよいよファイル接続です。inp オブジェクトと dat オブジェクトを、code オブジェクトに接続してみます。

```
>>> code.inputs([inp, dat])
>>>
```

特に何も変わった様子は見られないですが、実は code オブジェクトの内部では、個々のファイルの割り当てについて多くの処理が行われています。ここでは詳細は割愛しますが、code

オブジェクトの中を少し覗いてみましょう。少し煩雑ですが、各オブジェクト変数名やその内容と、図 A2-10 の内容を比較していけば、大まかな意味合いは理解できると思います。

```
>>> code.show()
===== MyFortCode  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = False
case =
comment <type 'dict'> = {1: 'Counting Data', 10: 'Counting Summry Data',
5: 'User Input Cards', 6: 'Editing List'}
contents =
data =
files <type 'dict'> = {1: <sagep.File.DataFile.DataFile instance at
0x403aaa0c>, 5: <sagep.File.CardFile.CardFile instance at
0x403aa9ec>}
fixed_path_flag <type 'bool'> = False
full_path_name = /users/test/MyFortCode
gen_code =
generated <type 'dict'> = {1: 'CODE1', 10: '', 5: '', 6: ''}
hint <type 'dict'> = {1: '', 10: '', 5: '', 6: ''}
is_temporary <type 'bool'> = False
mod_name = MYFORTCODE
mode <type 'dict'> = {1: 'i', 10: 'o', 5: 'i', 6: 'o'}
omittable <type 'dict'> = {1: False, 10: False, 5: False, 6: False}
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = True
restore_path =
target <type 'dict'> = {1: '*', 10: '*', 5: '*', 6: '*'}
target_code =
type <type 'dict'> = {1: 'COUNT_DATA', 10: 'COUNT_SUMMRY_DATA', 5:
'USER_CARD', 6: 'EDIT_LIST'}
===== MyFortCode  (END) =======
>>>
```

ここで特に注目すべきは、files というディクショナリ変数です。ここでは、ユニット番号とファイルの接続が期待通りに解決され、データファイルオブジェクトのインスタンスが保存されていることが分かります。

では次に、計算コードを実行してみましょう。

```
>>> code.run("test_case")
=== EXECUTING MYFORTCODE for the case of TEST_CASE
0
>>>
```

実行させるための run()メソッドには、計算ケース名を指定します。ここでは何でも構いませんが、仮に”test_case” としておきましょう。run()メソッドを起動すると、実行中というメッセージがでて、無事終了します。表示される数字は、エラーコードを示しており、0 は正常終了を意味します。

では次に、計算結果を取り出してみます。計算リスト(6 番ユニット)と出力データファイル(6 番以外のユニット)の取り出しにはそれぞれ list()メソッド、outfiles()メソッドを用います。lst オブジェクトとして取り出した段階では、データはオブジェクト内のデータとして存在しており、ファイルとして取り出せません。特定のファイルに明示的に書き込むためには、restore()メソッドを用います。restore()メソッドの引数に、出力先のパス名を指定します。

```
>>> lst = code.list()
>>> lst.show()
===== DataFile  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = True
case = TEST_CASE
contents =
data = ['   1  1  1¥n', '   2  1  2¥n', '   1  2  3¥n', '   2  2  4¥n', '
1  3  5¥n', '   2  3  6¥n']
fixed_path_flag <type 'bool'> = False
full_path_name =
gen_code = MYFORTCODE
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code = *
type = EDIT_LIST
===== DataFile  (END) =======
>>> lst.restore("/users/test/fort6.dat")
>>>
```

10 番ファイルの取り出しもほぼ同等です。実際にやってみましょう。

```
>>> out = code.outfiles()
>>> len(out)
1
>>> out[0].show()
===== DataFile  (BEGIN) =====
binary_flag <type 'bool'> = False
capture_flag <type 'bool'> = True
case = TEST_CASE
contents =
data = ['   1  1  1¥n', '   2  1  2¥n', '   1  2  3¥n', '   2  2  4¥n', '
1  3  5¥n', '   2  3  6¥n']
fixed_path_flag <type 'bool'> = False
full_path_name =
gen_code = MYFORTCODE
is_temporary <type 'bool'> = True
open_flag <type 'bool'> = False
property <type 'dict'> = {}
protect_flag <type 'bool'> = False
restore_path =
target_code = *
type = COUNT_SUMMRY_DATA
===== DataFile  (END) =======
>>> out[0].restore("/users/test/fort10.dat")
>>>
```

出力されたファイルを見てみると、期待通りの出力になっています。

```
% more /users/test/fort10.dat
   1  1  1
   2  1  2
   1  2  3
   2  2  4
   1  3  5
   2  3  6
%
```

以上が、MyFortCode を Python 言語から利用可能とするためのカプセル化の概要です。

A2-3.3　ファイル管理について

前節で示したように、入力カードや一般ファイルなどは、ファイルオブジェクトとして抽象化され、取り扱われていました。このファイルオブジェクトの管理には、FileManager クラスが便利に使えます。

FileManager は、一種のオブジェクトデータベースであり、次の機能を有します。なお、各メソッドの使用例については、テストコード(FileManagerTest.py)を参照してください。

- ファイルオブジェクトの追加（add メソッド）
- ファイルオブジェクトの抹消（delete メソッド）
- オブジェクト数の取得（size メソッド）
- 全オブジェクトの破棄（clear メソッド）
- ファイルオブジェクトの取りだし（outfiles, select メソッド）

ここでは、FileManager を使うメリットについて簡単に説明します。FileManager オブジェクトに対してファイルオブジェクトを登録(add)する際に、FileManager は以下の情報でオブジェクトのインデックス付けを行います：

(1) ファイルオブジェクトの計算した計算コード
(2) ファイルオブジェクトのファイルタイプ
(3) 計算ケース名

逆に、上記インデックスをキーにして選択的にオブジェクトを取り出す(select)ことも出来ます。例えば、同一計算コードを用いて多くの計算ケースを実行する場合、計算ケース名を適切に設定しておき、出力ファイルを FileManager に登録すれば良いでしょう。後に、特定ケースの出力ファイルのみを取り出したい場合には、当該ケース名をキーとして FileManager に問い合わせを行えば良いわけです。なお、Scenario クラスにおいては、このキーによる選択機能を活用することにより、FileManager をオブジェクトデータベースとして活用しています。

また、FileManager クラスは、SerializableAndVisible クラス(sagep.util モジュール)を継承しているため、オブジェクトの永続化にも対応しています。つまり、FileManager のインスタンスの保存・復元が行えます。これを活用することで、計算リスタート機能を容易に実現することが可能です。

A2-3.4　その他

既存コードの入力を簡便に取り扱いたい場合には、二通りの方法が考えられます。ひとつ目は、3.2 節で説明したように CardFile を用いる方法です。もう一つは InfoManager や InputGenerator を当該コード用に用意する方法です。小規模な計算コードの場合には前者で十分対応できるので、無理に後者のような複雑な方法を採る必要はないでしょう。

PSAGEP では、計算コードの全体挙動は、制御層の中心である Scenario クラスと、その上位に位置する Psagep クラスが管理しています。これは、PSAGEP では解析シナリオによって、コードに接続するファイルの種類や数が変わることがあるためです。計算コードの挙動が変わることのない小規模なケースの場合では、これらを明確に区別する必要はないでしょう。

既存コードをカプセル化することで、様々な応用が考えられます。表 A2-2 にはそれらの一例を示します。本フレームワークを活用し、是非ともあなたのアイデアを実現して下さい。

表 A2-2　PSAGEP カプセル化フレームワークの応用例

用途	説明
パラメータ サーベイ	複数の計算ケースに対応する入力を Python 側で自動生成し、計算ジョブを実行した後、FileManager でデータファイルを管理する。ポスト処理も Python で実装すれば、一貫したデータ管理が行える。
カップリング計算	抽象化されたデータファイルオブジェクトを用いて、複数の計算コードを連結する。制御層を Python で記述することができるため、アルゴリズム開発が簡便に行える。また、分散実行等も比較的容易に実装できる。
プロトタイプ実装	システム構築する際の API (Application Program Interface)の検討を行うためのプロトタイプ実装に活用する。既存コードをカプセル化したものをスタブとして利用して開発を行い、その後新規開発モジュールと置き換えることにより、ハイブリッド開発からスクラッチ開発へスムーズに移行出来る。