卒業論文

連続マルコフ過程モンテカルロ法と相関サンプリング法を用いた使用済燃料プールにおけるリスク評価の感度解析

名古屋大学
工学研究科
総合エネルギー工学専攻
エネルギー量子制御グループ
森下裕貴
令和 4 年 2 月

目次

1　序論

1.1　背景

2011 年 3 月 11 日に発生した東北地方太平洋沖地震により、東京電力福島第一原子力発電所事故（1F 事故）が発生した。1F 事故ではサイト内のプラントが同時多発的に被害を受け、事故が進展したことで事故対応が難化し、被害が拡大した。この事故対応の脆弱性が露呈した 1F 事故を契機として、複数の原子炉ユニット（マルチユニット）が関与するリスク評価について関心が高まった。

現在、原子炉施設を対象とするリスク評価では、確率論的リスク評価（Probabilistic Risk Assessment：PRA）が広く用いられている。特に、マルチユニットの影響を考慮したリスク評価では、事故シナリオ分岐確率の時間依存性と相互依存性を考慮可能である動的 PRA が注目されている。近年では、動的 PRA にシミュレーションを用いる方法として、連続マルコフ過程モンテカルロ（Continuous Markov chain Monte Carlo：CMMC）法を事故進展解析やプラントシミュレーションと組み合わせることで事故シナリオ発生頻度を定量化する手法（CMMC カップリング手法）が提案されている。CMMC カップリング手法によるリスク評価では、プラント状態遷移過程をマルコフ過程として扱い、各時刻におけるプラント状態をモンテカルロシミュレーションにより決定する。この方法では、初期乱数を変えた解析を多数回実施することにより、様々な事故シナリオを生成し、事故事象の発生頻度の定量化をすることが可能である。また、プラント状態遷移過程をマルコフ過程として扱うため、事故シナリオ分岐確率の評価において設備・機器の相互依存性や時間依存性を考慮することができる。

先行研究によって、CMMC 法を用いたマルチユニットの影響を考慮したリスク評価の妥当性が確認されている[1]。しかし、CMMC 法のリスク評価は動的マルチユニット PRA において効果的である一方で、①多くのシナリオを評価する必要がある、②計算時間が長い、③結果に統計誤差が発生するなどの課題もある。特に、計算結果に統計誤差が含まれることから、入力パラメータを摂動させて計算結果の変化を評価する単純な感度解析は、大きな統計誤差が伴うため実用的でない。例えば、ある安全系の故障率の変化が燃料損傷頻度に与える影響は、CMMC 法では評価が困難である。PRA 評価結果は、プラントの脆弱性の特定のために使われることが多く、そのためにはプラントの機器類の信頼性の変化などが事故のシナリオ、事故の確率、事故の影響に与える影響を評価することは重要である。以上のことから、CMMC 法によるリスク評価の枠組みにおける感度解析手法を検討する必要がある。

1.2　本研究の目的

本研究では、相関サンプリング法を用いて、CMMC 法における感度解析手法を検討することを目的とする。以下で述べる小目的を順に達成することで本研究の目的を達成する。

- CMMC 法への相関サンプリング法の適用の検討

相関サンプリング法は中性子の輸送計算におけるモンテカルロ計算に用いる手法であるが、これまで CMMC 法に相関サンプリング法を適用した例はない。そのため、CMMC 法に相関サンプリング法を適用する時のウエイトの導出とウエイトの適用法を検討する。この検討は第 2 章にて説明する。

- 相関サンプリング法を用いた単純なシステムに対する感度解析の妥当性確認

導出したウエイトを用いて単純なシステムに対する感度解析を行う。単純なシステム(一つの機器を持つシステム、二つの機器を持つシステム)へ適用し、解析解と比較することで本手法の妥当性を確認する。この検討は第 3 章にて説明する。

- 相関サンプリング法を用いた現実的なモデルを対象とした感度解析

現実的なモデルとして使用済燃料プールの解析モデルを作成し、相関サンプリング法と CMMC 法を用いて機器類の故障率に対する燃料損傷頻度の感度解析を行う。この検討は第 3 章にて説明する。

1.3　本論文の構成

本論文は、本文全 5 章及び Appendix 全 1 章の構成となっている。以下で、各章の概要を述べる。

- 第 1 章では、本研究の背景と目的を述べた。
- 第 2 章では、確率論的リスク評価の感度解析について述べる。初めに、確率論的リスク評価とその感度解析について説明する。次に、CMMC 法の概要について述べる。最後に、相関サンプリング法の理論について述べて、CMMC 法への相関サンプリング法の適用方法を説明する。
- 第 3 章では、単純なシステムを対象とした感度解析について述べる。第 2 章で説明した相関サンプリング法を用いて、一つの機器を持つシステムと二つの機器を持つシステムの感度解析を行う。解析解と比較することで CMMC 法と相関サンプリング法を用いた感度解析の妥当性を確認する。
- 第 4 章では、使用済燃料プールを対象とした感度解析について述べる。第 2 章で説明した相関サンプリング法を用いて、現実的なシステムとして使用済燃料プールを対象とした感度解析を行い、燃料損傷頻度の変化量を計算する。
- 第 5 章では、本研究のまとめと今後の課題を述べる。
- Appendix. A では、解析に用いた計算コードと簡単な説明をまとめる。

参考文献

[1] 澤田憲人, “マルチユニットリスク評価への連続マルコフ過程モンテカルロ法の適用,” 修士論文, 名古屋大学, (2021).

2　確率論的リスク評価の感度解析

2.1　本章の概要

本章では、連続マルコフ過程モンテカルロ（Continuous Markov chain Monte Carlo : CMMC）法と相関サンプリング法を用いた確率論的リスク評価（Probabilistic Risk Assessment : PRA）の感度解析手法についてまとめる。CMMC 法と相関サンプリング法を用いた PRA の感度解析を行う方法を理解するために、PRA の理論、CMMC 法の理論と PRA への適用、相関サンプリング法の理論と PRA の感度解析への適用について説明する。相関サンプリング法はもともと中性子輸送理論に用いられている手法であり、その手法をどのようにして CMMC 法を用いた PRA に適用したのか詳しく説明する。

本章の構成は以下の通りである。

- 2.1 節では、本章の概要をまとめる。
- 2.2 節では、確率論的リスク評価について述べる。PRA の理論を説明した後、PRA の感度解析について説明する。
- 2.3 節では、CMMC 法について述べる。CMMC 法の計算理論を説明した後、PRA に対してどのように適用するかを説明する。
- 2.4 節では、相関サンプリング法について述べる。相関サンプリング法の理論を説明し、CMMC 法を用いた PRA の感度解析への適用について説明する。相関サンプリング法による感度解析時に、用いるウエイトについての計算理論について説明する。
- 2.5 章では、本章のまとめを述べる。

2.2　確率論的リスク評価

2.2.1確率論的リスク評価の理論

本項では、確率論的リスク評価について述べる。

確率論的リスク評価（Probabilistic Risk Assessment : PRA）とは、原子力施設等で発生する様々な事故を対象として、そのシナリオ、発生頻度、発生時の影響を定量評価するアプローチである[1]。一般的にはイベントツリー法を用いた静的 PRA によるリスク評価が行われている。イベントツリーは一連の事象が時間とともにどのように分岐していくかを表現し、起因事象から起こりうる、様々な事柄を確認するための帰納法的論理手段を提供してくれる。

しかし、分岐確率が時間依存性を持つ場合や他の機器の状態に依存するシステムについては静的 PRA では取り扱いが難しくなる。複数の原子炉施設を考慮した事故は事故シナリオ分岐確率や時間依存性の考慮が必要であるため、これらが考慮可能である動的 PRA が注目されている。動的 PRA を行う手法として、シミュレーションを用いる方法がある。シミュレーションを用いる方法は、シミュレーションにより得られた温度・圧力と言ったパラメータを用いてシナリオの分岐確率を更新することで、より現実に近い解析結果を得ることが可能である。また、初期乱数を変更してシミュレーションを何度も実行することにより、

様々な事故シナリオを作成することが出来る。本論文で取り上げている CMMC 法はシミュレーションを用いる方法である。

ここではイベントツリーを用いた PRA を説明する。例として前日の雨による午後の試合の中止を考える。雨を起因事象として、午前中の太陽による蒸発、スポンジによる吸水、他グラウンドの確保を影響緩和系として想定し、試合の決行/中止をツリーで表現する。影響緩和系の午前中の太陽による蒸発、スポンジによる吸水、他グラウンドの確保の成功確率をそれぞれ、0.5、0.7、0.2 とするとイベントツリーは図 2-1 のようになる。

起因事象	事故緩和系システム			No.	試合	決行の発生頻度	中止の発生頻度
雨	午前中の太陽による蒸発	スポンジによる吸水	他グラウンドの確保			(/年)	(/年)
	0.5			1	決行	5	
年に10回	0.5	0.7		2	決行	3.5	
		0.3	0.2	3	決行	0.3	
			0.8	4	中止		1.2
					合計値	8.8	1.2

図 2-1　雨による試合の中止のイベントツリー(成功確率の向上なし)

このイベントツリーから、前日が雨だった試合(年 10 回を想定)のうち 1.2 回の試合が中止になることを算出できる。このようにして初期事象の発生頻度と事象の分岐確率を与えることにより、ツリーの中間や末端に現れる各事象の発生頻度を求めることができる。

次に、スポンジなど、グラウンドの水を排除する方法を効率化することで、吸水の成功確率を 0.7 から 0.8 に増加できるとすると、イベントツリーは図 2-2 のようになる。

起因事象	事故緩和系システム			No.	試合	決行の発生頻度	中止の発生頻度
雨	午前中の太陽による蒸発	スポンジによる吸水	他グラウンドの確保			(/年)	(/年)
	0.5			1	決行	5	
年に10回	0.5	0.8		2	決行	4	
		0.2	0.2	3	決行	0.2	
			0.8	4	中止		0.8
					合計値	9.2	0.8

図 2-2　雨による試合の中止のイベントツリー(スポンジによる吸水の成功確率の向上)

また、スポンジによる吸水の成功確率の向上ではなく、他グラウンドの確保に力を入れることで成功確率を 0.2 から 0.3 に増加できるとすると、イベントツリーは図 2-3 のようになる。

起因事象	事故緩和系システム			No.	試合	決行の発生頻度	中止の発生頻度
雨	午前中の太陽による蒸発	スポンジによる吸水	他グラウンドの確保			(/年)	(/年)
	0.5			1	決行	5	
年に10回	0.5	0.7		2	決行	3.5	
		0.3	0.3	3	決行	0.45	
			0.7	4	中止		1.05
					合計値	8.95	1.05

図 2-3　雨による試合の中止のイベントツリー(他グラウンドの確保の成功確率の向上)

図 2-2、図 2-3 から、スポンジによる吸水の成功確率が向上すると試合中止の発生頻度が年に 0.8 回、他グラウンドの確保の成功確率が向上すると試合中止の発生頻度が年に 1.05 回になることが計算でき、初期状態の 1.2 回より減少することが分かる。このことから、イベントツリーはある対策を追加することで、その対策の影響によりどの程度発生頻度を低減できるか確認することができるといえる。

2.2.2確率論的リスク評価の感度解析

本項では、確率論的リスク評価の感度解析について述べる。

感度解析とは、入力を変化させたとき、出力にどのくらい影響を与えるかを定量的に評価することである。本論文では感度解析に重きをおいて解析を行うが、ここでは他の評価方法としてリスク重要度指標を用いるリスク評価について説明する。

リスク重要度指標とは、リスク情報の活用に有用な定量的情報のことであり、システム内の支配的な因子を特定する際に有用である[2]。前項で PRA には、イベントツリーを用いる方法とシミュレーションを用いる方法があると説明したが、イベントツリーを用いる方法では、イベントツリーに与える分岐確率を変化させる形で、CMMC 法では、機器の故障確率を変化させる形でリスク重要度指標を計算ことが出来る。

代表的なリスク重要度指標として、リスク増加価値（Risk Achievement Worth：RAW）と FV（Fussell-Vesely）重要度を説明する。出力パラメータとして炉心損傷頻度を例にとる。

- リスク増加価値（Risk Achievement Worth：RAW）

  対象としている安全設備等が故障した場合、炉心損傷頻度がどれだけ増加するかを示す指標であり、次式のように定義される。

$$\mathrm{RAW} = \frac{F(CD|A=1)}{F(CD)} \tag{2.1}$$

  ここで、

  $F(CD|A=1)$：事象 A(注目している安全設備などの故障)の生起確率が 1 の場合の炉心損傷頻度

$F(CD)$：炉心損傷頻度

RAW が 1 に近い値の場合、その機器の安全への影響は小さいと判断できる。そのため、RAW は当該系統または機器を除外した場合のリスクの影響把握に有効とされる。

- FV（Fussell-Vesely）重要度

  対象としている安全設備等の故障が発生しない場合を仮定した時の炉心損傷頻度への寄与割合を示す指標であり、次式のように定義される。

$$\begin{aligned} \mathrm{FV} &= \frac{F_A(CD)}{F(CD)} \\ &= \frac{F(CD) - F(CD|A=0)}{F(CD)} \end{aligned} \tag{2.2}$$

  ここで、

  $F_A(CD)$: 事象 A(対象としている安全設備等の故障)が寄与して発生する炉心損傷頻度

  $F(CD)$：炉心損傷頻度

  $F(CD|A=0)$：事象 A の生起確率が 0 の場合の炉心損傷頻度

  特定の機器の故障、人的過誤の発生確率を低減することにより、どれほどの安全性の向上が望めるか判断できる。FV 重要度は、当該系統または機器の故障率等の低減による信頼性向上効果の把握に有効とされる。

二つのリスク重要度指標を紹介したが、どちらも事象 A の生起確率を 0 か 1 としていて、極端な影響のみを考慮しているといえる。

2.3　連続マルコフ過程モンテカルロ法

2.3.1連続マルコフ過程モンテカルロ法の理論

本項では、連続マルコフ過程モンテカルロ（CMMC）法の理論をまとめる[3]。

連続マルコフ過程モンテカルロ法（マルコフ連鎖モンテカルロ（MCMC）法とも呼ばれる。こちらの呼び方の方が浸透していると思われる。）は、1990 年代に統計学の分野で開発されたモンテカルロ法の一種である。CMMC 法を理解するためには、マルコフ過程とモンテカルロ法を理解する必要があり、本項で説明する。

マルコフ過程とは、ある対象の未来の状態が、過去の状態によらず現在の状態にのみ依存するとする確率過程である。つまり、現在のパラメータのみから次の時間ステップである未来のパラメータを算出することになり、数列の漸化式のようなイメージとして捉えることが出来る。連続マルコフ過程とは、マルコフ過程を時間的に連続的に取り扱うことを意味する。

モンテカルロ法とは、乱数を用いた数値シミュレーション手法の総称であり、原子炉物理の分野では、輸送理論や拡散理論などに基づき、空間・角度・エネルギーを離散化して計算

する決定論的手法に対して、確率論的手法と呼ばれる。原子炉物理の分野では、モンテカルロ法は近似の少なさからよく参照解として用いられる。モンテカルロ法では、現象をモデル化した数式を解く際に乱数を使用し、得られた複数の結果を統計的に処理することで解を算出する。

以上を踏まえると、CMMC 法とは現在の状態によって遷移確率が決まり、なおかつ乱数によって未来の状態の遷移を決定する手法である。

2.3.2確率論的リスク評価への適用

本項では、CMMC 法を用いた PRA 手法の一例として、CMMC カップリング手法の概要について述べる。

CMMC カップリング手法とは、CMMC 法とプラント状態解析コードや事故進展解析コードなどと組み合わせることで、プラント挙動と共に事故進展を解析する手法である。プラントの初期状態から最終状態まで解析することにより、事故シナリオを生成することができる。CMMC 法では、乱数を使用して計算を行うため、初期乱数を更新することで様々な事故シナリオを生成することができる。CMMC 法とプラント状態解析を組み合わせた CMMC カップリング手法による事故シナリオ生成の概略を図 2-4 に示す。

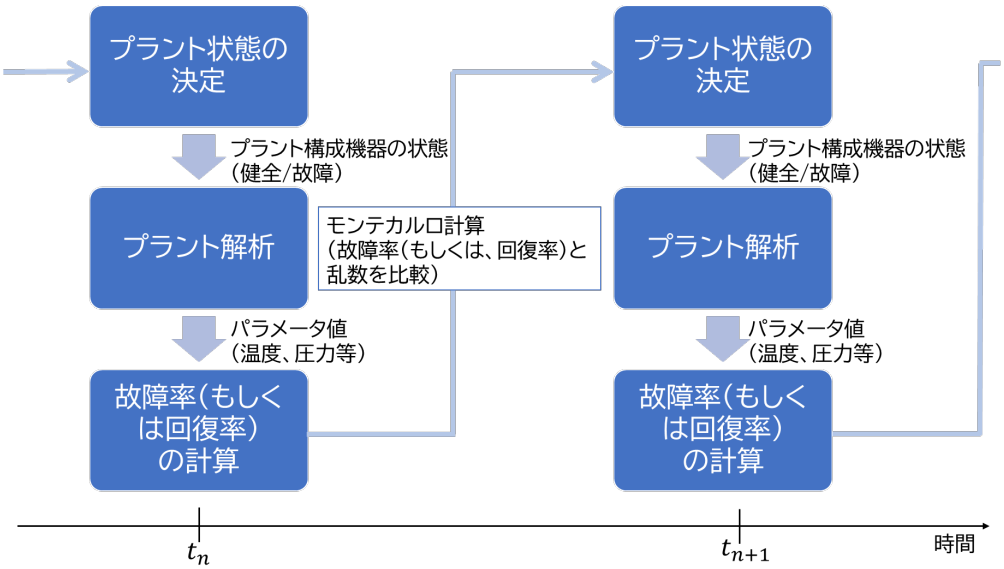


図 2-4　CMMC カップリング手法における事故シナリオの生成

具体的なシナリオ生成を説明する。まず、プラント状態から、事故進展解析コードなどのシミュレーションによりプラント解析を行い、ある時刻におけるパラメータ（温度、圧力、応力、流量、重量など）を得る。次に、解析で得られたパラメータを用いて、その時刻におけるプラント構成機器の故障率（もしくは回復率）を計算する。さらに、計算された故障率と生成された乱数を比較することで、プラント構成機器の状態を決定し、次のステップでの

プラントシステムの状態を決定する。このプラント状態を用いて、次の時間ステップでのプラント状態をプラント解析により予測する。

以上の 3 ステップで構成されるサイクルを解析終了まで繰り返し行うことで、1 つの事故シナリオが形成される。モンテカルロ法における初期乱数を変化させることで、試行回数分の事故シナリオを生成することが出来る。多数生成されたシナリオを統計的に処理することで、事故シナリオが発生する確率とその信頼区間などの情報を得ることが可能となる。

なお、本論文では CMMC カップリング手法を単に CMMC 法と呼ぶことが多い。

2.4　相関サンプリング法

2.4.1相関サンプリング法の理論

本項では、相関サンプリング法の理論をまとめる。

相関サンプリング法は、中性子輸送の分野で用いられている手法であり、モンテカルロ法において入力が微小変化する際の計算（摂動計算）を可能とすることを目的として開発されたものである。相関サンプリング法は、昔からある手法で、1960 年代に統計学の分野ですでに開発されている[4], [5]。

ここで、中性子輸送における相関サンプリング法について説明する。一例として、体系における巨視的断面積を摂動させる場合を考える。また、飛行については連続的で、かつ状態が飛行前と飛行後の二つある場合について考える。一回の飛行をするときの中性子輸送の基準体系と摂動体系のイメージを図 2-5 に示す。

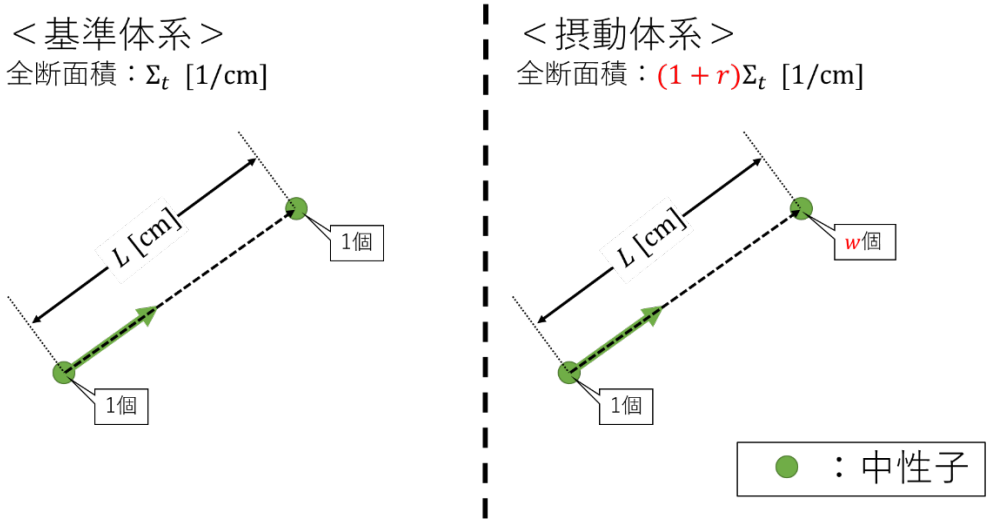


図 2-5　中性子輸送の基準体系と摂動体系（飛行のみ）

直線で距離L [cm]だけ中性子が飛ぶと仮定し、基準体系の断面積がΣ_t [1/cm]、摂動体系の断面積を$\Sigma_t(1+r)$ [1/cm]とする。r [−]は、摂動割合とよばれるパラメータである。また、

飛行前の中性子束をψ_0とし、基準体系と摂動体系での飛行後の中性子束をそれぞれψ_{base}, ψ_{per}とすると解析解は次式のようになる。

$$\begin{aligned} \psi_{base} &= \psi_0 \times \exp(-\Sigma_t L) \\ \psi_{per} &= \psi_0 \times \exp(-(1+r)\Sigma_t L) \end{aligned} \tag{2.3}$$

式**(2.3)**より、基準体系と摂動体系の中性子が距離L [cm]だけ通り抜ける確率は次式のようになる。

$$\begin{aligned} \frac{\psi_{base}}{\psi_0} &= \exp(-\Sigma_t L) \\ \frac{\psi_{per}}{\psi_0} &= \exp(-(1+r)\Sigma_t L) \end{aligned} \tag{2.4}$$

式**(2.4)**で求めた$\frac{\psi_{base}}{\psi_0}, \frac{\psi_{per}}{\psi_0}$からウエイト$w$を次のように定義する。

$$\begin{aligned} w &= \frac{\frac{\psi_{per}}{\psi_0}}{\frac{\psi_{base}}{\psi_0}} \\ &= \frac{\psi_0 \times \exp(-(1+r)\Sigma_t L)}{\psi_0 \times \exp(-\Sigma_t L)} \\ &= \exp(-r\Sigma_t L) \end{aligned} \tag{2.5}$$

ここで定義したウエイトwは基準体系の中性子束に比べて摂動体系の中性子がどのくらい変化するかを表すパラメータとなる。具体的には、摂動体系において、中性子が基準体系と同一の飛行経路をたどったときに、中性子距離L [cm]だけ通り抜ける確率の比(摂動体系/基準体系)を表すものとなっている。従って、ウエイトが計算できれば、モンテカルロ法で基準体系の中性子を求めるだけで、摂動体系のパラメータを同時に評価することができる。別の言い方をすれば、摂動体系の中性子の挙動を別途モンテカルロ計算で直接求める必要がなくなるということになる。

中性子輸送の摂動モンテカルロ法のフローチャートは図 2-6 のようになる。例えば、ウエイトwが 2 の時、摂動体系の中性子数は基準体系の中性子数の 2 倍となり、ウエイトwが 0.5 の時、摂動体系の中性子数は基準体系の中性子数の 0.5 倍となる。

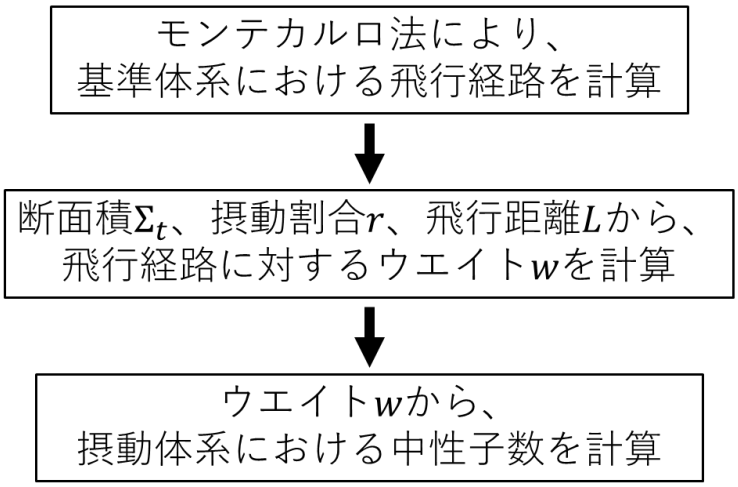


図 2-6　中性子輸送における摂動モンテカルロ計算

以上から、相関サンプリング法は、モンテカルロ計算の基準体系の結果とウエイトから摂動体系の結果を求めることができる手法といえる。なお、基準体系と摂動体系では中性子の平均自由行程が変化するが、相関サンプリング法では、中性子の飛行経路が基準体系と摂動体系で変化しないと仮定する。

2.4.2感度解析への適用

本項では、CMMC 法を用いた PRA に対する、相関サンプリング法による感度解析について説明する。

CMMC 法は 2.3 節で述べた通り、現在の状態によって遷移確率が決まり、乱数によって遷移が決定する手法である。現在の状態に注目すれば、中性子輸送での適用例に倣って、適用することが可能である。ここでは、存在確率に注目して説明する。存在確率が分かれば累積故障確率も分かるからである。以下では、機器の故障率を摂動させる場合を考える。

計算手順は以下の通りである。

① CMMC 法により基準状態における次のタイムステップでのシステムの状態を計算する。ここで、システムの状態とは、複数の機器の状態(健全/故障)の組み合わせから決まるものである。
② 機器の故障率λや故障率の摂動割合r、時間幅Δtからウエイトwを更新する。
③ ウエイトwから、摂動体系におけるシステムの状態を計算する。
④ ①～③を考慮する事故シナリオの計算時間だけ繰り返し計算することで、摂動体系における事故シナリオを得る。

ここで、存在確率は現在の状態のみで決まっておらず、過去の状態の積分で決まる。したがって、摂動体系の存在確率P_{per}を求めるためのウエイトについても、現在の状態のみでは決

まらないことに留意が必要である。相関サンプリング法と CMMC 法を用いた感度解析のフローチャートは図 2-7 のようになる。

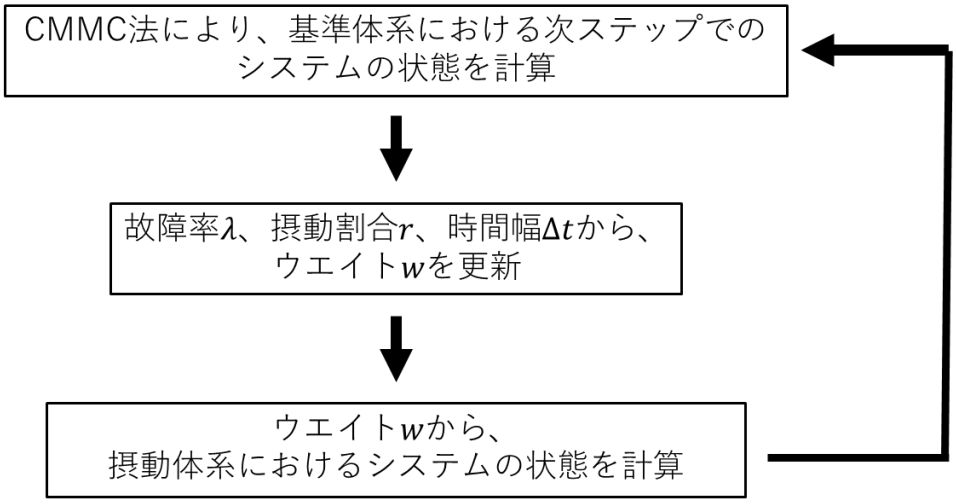


図 2-7　PRA における摂動モンテカルロ法

2.4.3相関サンプリング法に用いるウエイト

本項では、事故シナリオに固有のウエイトについて説明する。

相関サンプリング法を CMMC 法に適用するためには、注目する要素(機器の状態)が変化することを考慮する必要がある。そこで、事故シナリオに固有のウエイトを考える導入として、中性子輸送計算におけるか相関サンプリング法の適用を例に取り説明する。

2.4.1 項で説明した中性子輸送での相関サンプリング法は、中性子が衝突して消滅することを仮定していたが、ここでは、衝突後に中性子の状態(エネルギー)が変化することを考慮に入れられるように拡張する。具体的には、高速中性子が原子核との散乱によって減速し熱中性子になる変化を追加する。また、説明を簡単にするため、全断面積が散乱断面積と同じであると仮定し、高速中性子が原子核と相互作用を起こすと必ず熱中性子に変化すると仮定する。ここでも、1 つの中性子に着目する。高速中性子の散乱により熱中性子になるイメージを図 2-8 に示す。なお、ここでの説明は、プラント解析において、機器の状態が変化することに対応する。

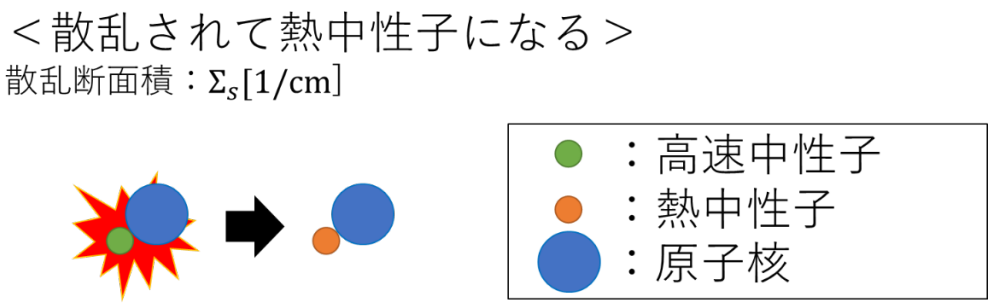


図 2-8　高速中性子の原子核による散乱

まずは、中性子が距離L [cm]だけ進んだ後、高速中性子が原子核との散乱によって減速し熱中性子になるときを考える。その飛行経路をたどる様子を図 2-9 に示す。

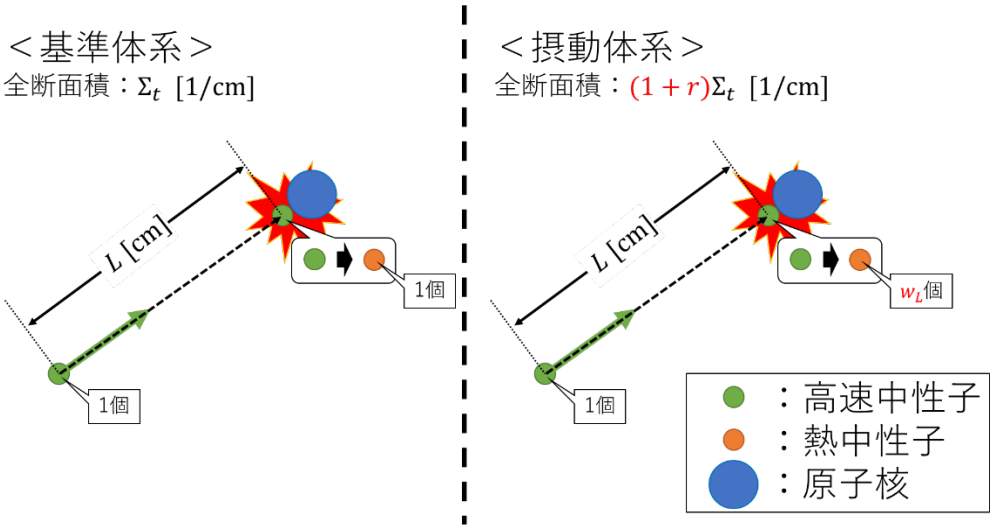


図 2-9　中性子輸送の基準体系と摂動体系（飛行→散乱）

図 2-9 で示したウエイトw_Lを計算する。まず、中性子が微少距離ΔL [cm]内で散乱する確率は基準体系と摂動体系それぞれ次式のようになる。

$$\frac{d\psi_{base}}{\psi} = \Sigma_t \Delta L$$
$$\frac{d\psi_{per}}{\psi} = (1+r)\Sigma_t \Delta L \tag{2.6}$$

次に、式(2.4)と式(2.6)から、中性子が距離L [cm]だけ通り抜けた後、微小距離ΔL [cm]内で散乱する確率は基準体系と摂動体系それぞれ次式のようになる。

$$\frac{\psi_{base}}{\psi_0} = \exp(-\Sigma_t L) \times \Sigma_t \Delta L$$
$$\frac{\psi_{per}}{\psi_0} = \exp(-(1+r)\Sigma_t L) \times (1+r)\Sigma_t \Delta L \tag{2.7}$$

よって、中性子が距離L [cm]だけ通り抜けた後、距離ΔL [cm]で散乱する確率の比(摂動体系/基準体系)は次式のようになる。

$$
\begin{aligned}
w_L &= \frac{\frac{\psi_{per}}{\psi_0}}{\frac{\psi_{base}}{\psi_0}} \\
&= \frac{\exp(-(1+r)\Sigma_t L) \times (1+r)\Sigma_t \Delta L}{\exp(-\Sigma_t L) \times \Sigma_t \Delta L} \\
&= \exp(-r\Sigma_t L)\,(1+r)\Sigma_t
\end{aligned}
\tag{2.8}
$$

このようにして求めたウエイトw_Lは、基準状態において中性子が一つでかつ一回の飛行のみを考えているため、摂動体系における熱中性子の個数と捉えることが出来る。また、違う見方をすればウエイトw_Lは、摂動体系において中性子が距離L [cm]だけ衝突せずに通り抜けた後、微小距離ΔL [cm]内で散乱する飛行経路をたどる数とも捉えることが出来る。飛行経路が複雑となる場合には、後者の捉え方が非常に重要となる。

次に中性子が二つで一つのセットと考える時のウエイトの計算について説明する。ことき、ウエイトは個々の中性子に対して計算されるのではなく、セットに対して計算される。シナリオとしては、二つの高速中性子が飛行し、二つの高速中性子のうち一つが先に散乱によって熱中性子に変化する。その後、熱中性子と高速中性子が飛行し、残りの高速中性子が散乱によって熱中性子に変化し両方とも熱中性子となる飛行経路を考える。また、説明を簡単にするため、全断面積が散乱断面積と同じであると仮定し、高速中性子が原子核と相互作用を起こすと必ず熱中性子に変化すると仮定する。さらに、散乱後の飛行方向は変わらないとし、散乱して熱中性子に減速した後は、熱中性子は原子核と相互作用を起こさないと仮定する。以上で説明したような飛行経路をたどる様子を図 2-10 に示す。

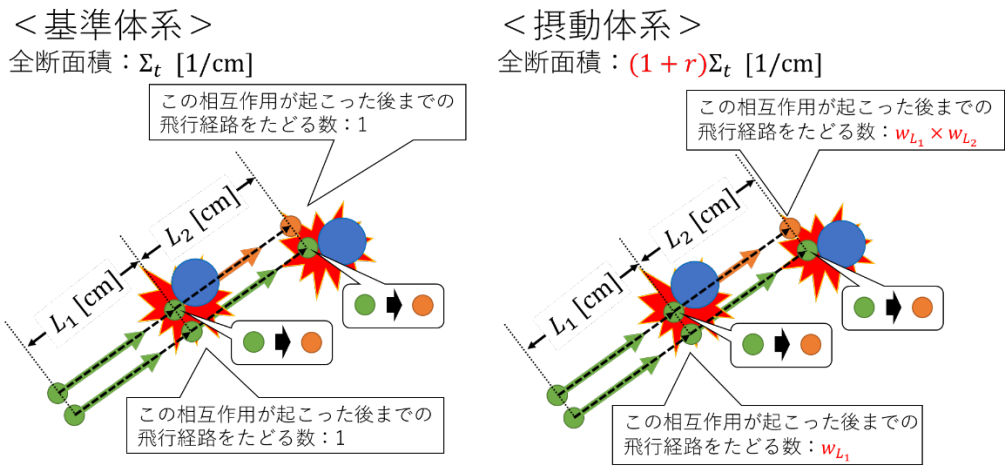


図 2-10　中性子二つをセットとした中性子輸送の基準体系と摂動体系（飛行経路に対するウエイトの評価）

図 2-10 で示したウエイトw_{L_1}、w_{L_2}を計算する。初期位置からL_1 [cm]進むまでのセットのウエイトw_{L_1}は「一つの中性子がL_1で衝突かつ二つの目の中性子がL_1まで衝突しない」状態に対して計算される。従って、L_1で衝突する一つ目の中性子のウエイトとL_1で衝突しない二つ目の中性子のウエイトの積で与えられ、式(2.4)と式(2.6)から、次式のように計算できる。

$$\begin{aligned} w_{L_1} &= \frac{\exp(-(1+r)\Sigma_t L_1) \times (1+r)\Sigma_t \Delta L}{\exp(-\Sigma_t L_1) \times \Sigma_t \Delta L} \times \frac{\exp(-(1+r)\Sigma_t L_1)}{\exp(-\Sigma_t L_1)} \\ &= \exp(-r\Sigma_t L_1)\,(1+r) \times \exp(-r\Sigma_t L_1) \\ &= \exp(-(r\Sigma_t + r\Sigma_t)L_1) \times (1+r) \end{aligned} \tag{2.9}$$

次にL_1 [cm]の位置からL_2 [cm]進むまでのウエイトw_{L_2}は、「熱中性子になった一つ目の中性子がL_2まで衝突せず、二つ目の中性子がL_2で衝突し熱中性子になる」状態に対して計算される。熱中性子になった後、原子核との相互作用を起こさない($\Sigma_t = 0$)ことに注意すれば一つ目の中性子のウエイトの変化はなく(1 のままであり)、ウエイトの変化が生じるのは二つ目の中性子においてのみである。従って、次式のようになる。

$$\begin{aligned} w_{L_2} &= \frac{\exp(-(1+r) \times 0 \times L_2)}{\exp(-0 \times L_2)} \times \frac{\exp(-(1+r)\Sigma_t L_2) \times (1+r)\Sigma_t \Delta L}{\exp(-\Sigma_t L_2) \times \Sigma_t \Delta L} \\ &= \exp(-r\Sigma_t L_2) \times (1+r) \end{aligned} \tag{2.10}$$

最後に、初期位置からL_2 [cm]進むまでのウエイト$w_{L_1+L_2}$はw_{L_1}、w_{L_2}の積で書くことが出来る。

$$\begin{aligned} w_{L_1+L_2} &= w_{L_1} \times w_{L_2} \\ &= \big(\exp(-(r\Sigma_t + r\Sigma_t)L_1) \times (1+r)\big) \times \big(\exp(-r\Sigma_t L_2) \times (1+r)\big) \end{aligned} \tag{2.11}$$

以上の流れで、飛行経路に対するウエイトを計算する。事故シナリオに固有のウエイトに対しても、同じようなイメージで計算する。飛行経路に対するウエイトの計算から事故シナリオのウエイトの計算にスムーズに移るために、二つのウエイトで使用する語句の対応を表 2-1 に示す。

表 2-1　中性子輸送と事故に用いる語句の対応表

中性子輸送	事故
中性子	機器
高速中性子	起動している機器
熱中性子	起動していない機器
衝突	故障
断面積	故障率
距離	時間
飛行経路	事故シナリオ

以下で使用する用語とその説明を表 2-2 で示す。

表 2-2　状態の存在確率に関する用語集

用語	用語の説明
故障率：$\lambda_{fail}(t)$	機器が健全であるとき、時刻tにおいて単位時間あたりに故障する確率
回復率：$\lambda_{rec}(t)$	機器が故障しているとき、時刻tにおいて単位時間あたりに機能が回復する確率
状態遷移確率：$p_{x\to y}(t)$	時刻tにおいて単位時間あたりに状態xから状態yに遷移する確率
存在確率：$P_x(t)$	時刻tにおいて状態xが存在する確率
故障率の摂動割合：$r_{fail}(t)$	故障率の摂動割合
回復率の摂動割合：$r_{rec}(t)$	回復率の摂動割合
時間幅：Δt_i	i 番目の時間幅

ここで、摂動を考えた時の故障率と摂動割合はそれぞれ$(1+r_{fail})\lambda_{fail}$、$(1+r_{rec})\lambda_{rec}$となる。また、システムの状態遷移図の一例を図 2-11 に示す。状態 a,b,c,d,e,x,yはシステム内の要素 A,B,C,…の状態によって定められる。また、状態xから状態yへの遷移に注目している。さらに、状態 a,b,c は状態xに遷移する可能性がある状態であり,状態 d,e は状態xから遷移してくる可能性がある状態である。

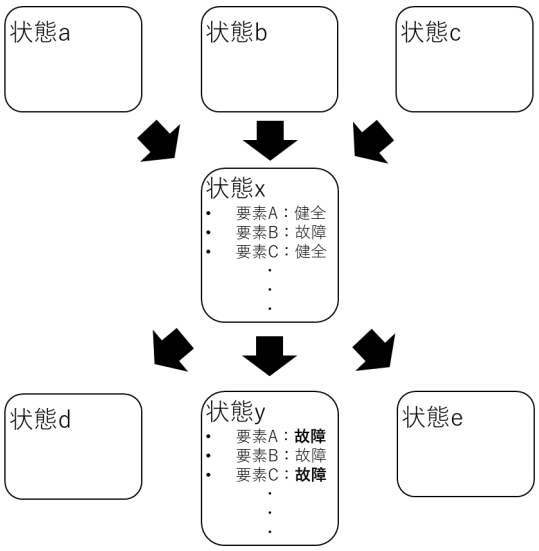


図 2-11　状態xに注目した時の一般的なシステムの状態遷移図

状態xに注目したとき、状態xから状態yに変化する以前の状態xの存在確率P_xは次式を満

たす。一行目で表す式が厳密な式であるが、二つ以上の故障率の積の項は非常に小さい値であるとし、近似的に 0 とした。近似を適用した式が二行目の式にあたる。

$$\begin{aligned} -\frac{dP_x}{dt} &= P_x \times (1-(1-\lambda_A)(1-\lambda_B)\cdots) \\ &\approx P_x \times (\lambda_A + \lambda_B + \cdots) \\ &= P_x \times \sum_{X \in x} \lambda_X \end{aligned} \tag{2.12}$$

上式において、一行目で表す式が厳密な式であるが、二つ以上の故障率の積の項は非常に小さい値であるとし、近似的に 0 とした。近似を適用した式が二行目の式にあたる。

時間Tの間、状態xに滞在する時の存在確率は、式(2.12)より次式のように表すことが出来る。

$$\begin{aligned} -\frac{dP_x}{dt} &= P_x \times \sum_{X \in x} \lambda_X \\ \int_1^{P_x} \frac{1}{P_x} dP_x &= \int_0^T \left(-\sum_{X \in x} \lambda_X \right) dt \\ P_x &= \exp\left(-\sum_{X \in x} \lambda_X T \right) \end{aligned} \tag{2.13}$$

次に、状態xから状態yに遷移する時の状態遷移確率は状態xの存在確率P_xと故障率（もしくは回復率）λを使って次式のように表すことが出来る。

$$p_{x \to y} = P_x \times \prod_{X \in x \to y} \lambda_X \, \Delta t \tag{2.14}$$

状態xから状態yの遷移は、関与する機器$(X \in x \to y)$のみが同時に故障(もしくは回復)することで発生する。そのため、$\prod_{X \in x \to y} \lambda_X$は故障率（もしくは回復率）$\lambda$を掛け合わせる。また、厳密には状態$x$から状態yの遷移に関与していない故障率（もしくは回復率）が発生しない確率を乗じる必要があるが、故障率（もしくは回復率）λは十分小さいと仮定しているため、$(1-\lambda) \approx 1$とし、明示的に乗じていない。

そして、基準体系の状態遷移確率と摂動体系の状態遷移確率を用いて、ウエイトwは次のように定義する。

$$w = \frac{p_{per}}{p_{base}} \tag{2.15}$$

式(2.13)と式(2.14)と式(2.15)から、時間Tの間状態xに滞在したのち、状態yに遷移する時、ウエイト$w_{x \to y}$は次式のようになる。

$$
\begin{aligned}
w_{x\to y} &= \frac{p_{x\to y,per}}{p_{x\to y,base}} \\
&= \frac{P_{x,per}\times\prod_{x\to y}(1+r_X)\lambda_X\,\Delta t}{P_{x,base}\times\prod_{x\to y}\lambda_X\,\Delta t} \\
&= \frac{\exp(-\sum_{X\in x}(1+r_X)\lambda_X\,T)}{\exp(-\sum_{X\in x}\lambda_X\,T)}\times\prod_{X\in x\to y}(1+r_X) \\
&= \exp\left(-\sum_{X\in x}r_X\lambda_X\right)T\times\prod_{X\in x\to y}(1+r_X)
\end{aligned}
\tag{2.16}
$$

ここで、一つのシナリオを例として挙げ、その時のウエイトの時間変化を表 2-3 に示す。状態は状態a, b, c, dがあるときで、機器は機器 A、B があるときを考える。初期時刻に状態a、時刻t_2に機器 A が故障して状態b、時刻t_4に機器 A が回復して状態a、時刻t_5に機器 A、B が同時に故障して状態cになっており、時刻t_7で計算時刻となり終了するシナリオを考える。そのシナリオ時のウエイトの時間変化を表 2-3 で示す。計算には式**(2.16)**を用いる。総乗Πは空集合であるとき 1 となるため、状態が遷移しない場合であっても用いることが出来る。

表 2-3　ウエイトの時間変化の一例（状態a（初期時刻）→状態b（時刻t_2）→状態a（時刻t_4）→状態c（時刻t_5）、時刻t_7で終了）

	w_t
t_0	1
t_1	$w_{t_0} \times \left(\exp\left(-\left(r_{A,fail}(t_0)\lambda_{A,fail}(t_0) + r_{B,fail}(t_0)\lambda_{B,fail}(t_0)\right)\Delta t_0\right) \times 1\right)$
t_2	$w_{t_1} \times \left(\exp\left(-\left(r_{A,fail}(t_1)\lambda_{A,fail}(t_1) + r_{B,fail}(t_1)\lambda_{B,fail}(t_1)\right)\Delta t_1\right) \times \left(1 + r_{A,fail}(t_1)\right)\right)$
t_3	$w_{t_2} \times \left(\exp\left(-\left(r_{A,rec}(t_2)\lambda_{A,rec}(t_2) + r_{B,fail}(t_2)\lambda_{B,fail}(t_2)\right)\Delta t_2\right) \times 1\right)$
t_4	$w_{t_3} \times \left(\exp\left(-\left(r_{A,rec}(t_3)\lambda_{A,rec}(t_3) + r_{B,fail}(t_3)\lambda_{B,fail}(t_3)\right)\Delta t_3\right) \times \left(1 + r_{A,rec}(t_3)\right)\right)$
t_5	$w_{t_4} \times \left(\exp\left(-\left(r_{A,fail}(t_4)\lambda_{A,fail}(t_4) + r_{B,fail}(t_4)\lambda_{B,fail}(t_4)\right)\Delta t_4\right) \times \left(1 + r_{A,fail}(t_4)\right)\left(1 + r_{B,fail}(t_4)\right)\right)$
t_6	$w_{t_5} \times \left(\exp\left(-\left(r_{A,rec}(t_5)\lambda_{A,rec}(t_5) + r_{B,rec}(t_5)\lambda_{B,rec}(t_5)\right)\Delta t_5\right) \times 1\right)$
t_7	$w_{t_6} \times \left(\exp\left(-\left(r_{A,rec}(t_6)\lambda_{A,rec}(t_6) + r_{B,rec}(t_6)\lambda_{B,rec}(t_6)\right)\Delta t_6\right) \times 1\right)$

2.4.4システムの状態に対するウエイト

本項では、システムの状態に対するウエイトの計算方法について説明する。

相関サンプリング法を CMMC 法に適用するためには、事故シナリオ固有のウエイトを求め、それを統計的に処理することでシステムの状態に対するウエイトも計算する必要がある。システムの状態とはシステムが含んでいる機器の状態に依存するもので、含む機器がm個、機器の状態がn種類ある場合、システムの状態はn^m通りあることになる。そのため、ある一つの状態を基準としたとき最大で$n^m - 1$通りだけ遷移の可能性がある。このシステムの状態の遷移には分岐が発生することから、シナリオのウエイトを用いてn^mだけの状態のウエイトを計算することになる。

まず、中性子輸送の場合で説明する。図 2-10 で考えた二つの中性子の輸送と同じシナリオを考える。この時の中性子のセットに対するウエイトを考える。この時の中性子輸送の基準体系と摂動体系のイメージをそれぞれ、図 2-12、図 2-13 に示す。

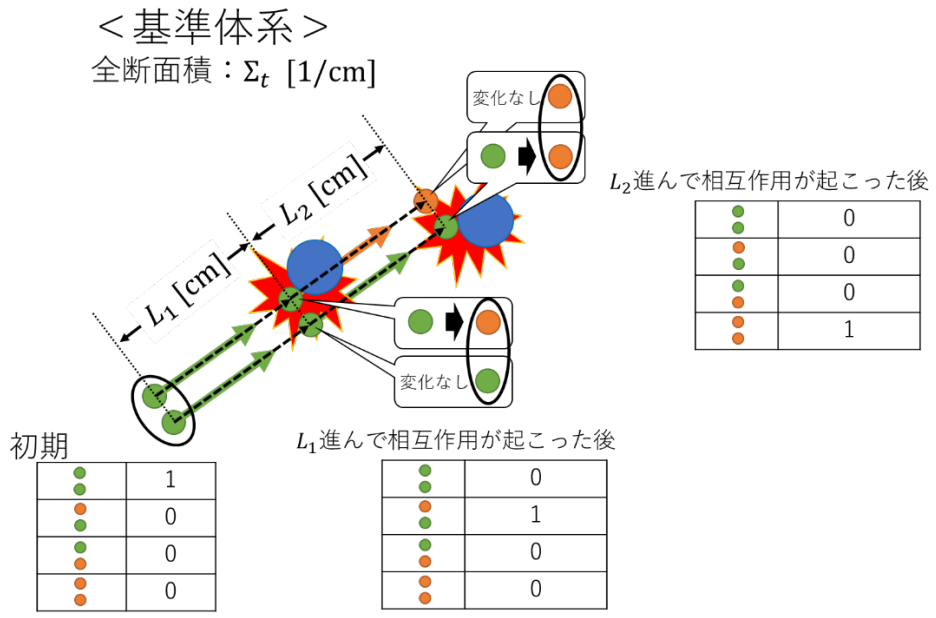


図 2-12　中性子二つをセットとした中性子輸送の基準体系
（中性子のセットに対するウエイトの評価）

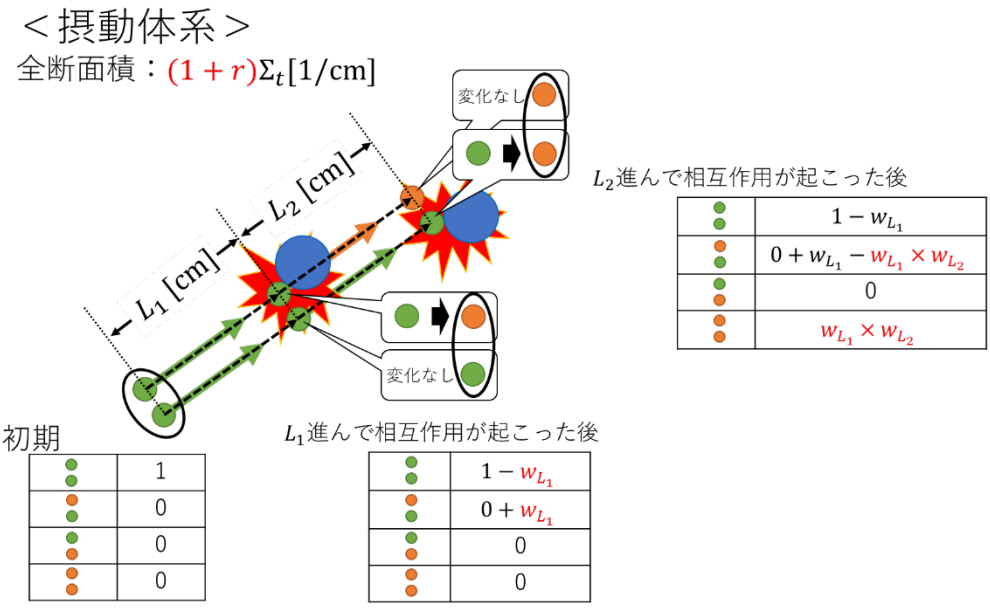


図 2-13　中性子二つをセットとした中性子輸送の摂動体系
（中性子のセットに対するウエイトの評価）

図 2-13 で用いているウエイトw_{L_1}、w_{L_2}は図 2-10 で用いているウエイトw_{L_1}、w_{L_2}と同じである。基準体系ではその飛行距離での中性子の組み合わせに 1、それ以外の中性子の組み合わせには 0 となる。一方、摂動体系では、中性子の組み合わせが変化したときに、変化先

にその飛行距離Lでのウエイトw_Lを加え、変化前にその飛行距離Lでのウエイトw_Lだけ減らす。また、中性子の変化に関わっていない中性子の組み合わせに対してはウエイトw_Lを作用させる必要はない。

ウエイトw_{L_1}が 1 より大きい場合、高速中性子の組み合わせのウエイト$1-w_{L_1}$は 0 よりも小さい値となり、高速中性子と熱中性子の組み合わせのウエイト$0+w_{L_1}$は 1 よりも大きい値となる。これは、断面積の摂動により距離L_1で一つの高速中性子が熱中性子になる確率が上昇し、高速中性子の組み合わせは基準体系より出来にくくなり、高速中性子と熱中性子の組み合わせは基準体系より出来やすくなることを意味している。

次に、システムの状態に対するウエイトの積算方法について説明する。ウエイトの積算方法は、表 2-4 に示す組み合わせが存在する。作用させるウエイトwはその時刻tでのウエイトw_tである。

表 2-4　ウエイトの作用のさせ方

	状態xのウエイト	状態yのウエイト	状態x, y以外のウエイト
状態xから状態yへの遷移時	$-w_t$	$+w_t$	± 0
状態xでの滞在時	± 0	± 0	± 0

初期の状態には$w=1$が、その他の状態には$w=0$が与えられている。例えば、上記の例において、初期状態がxである場合、計算開始時には$w_x=1$、$w_y=0$となる。CMMC 法の計算に伴い、表 2-1 に従って、各状態におけるウエイトが積算される形となっている。基準体系において、ある状態の存在確率P_{base}であるとする。ウエイトは、摂動体系においてその存在確率がどの程度変化するのかを表すものであることから、ウエイトとP_{base}を乗じることによって、摂動体系の存在確率P_{per}が得られる。

表 2-4 について説明する。説明を簡単にするため、その時刻でのウエイトw_tは単調増加[単調減少]する時を考える。この時考えるシナリオは故障率の摂動によって、基準体系より辿りやすい[辿りにくい]シナリオを考えているということになる。

- 状態xから状態yへの遷移時の状態yのウエイト

  摂動によって状態yに遷移しやすく[遷移しにくく]なっているため、遷移先の存在確率は大きく[小さく]なる。そのため、その時刻での状態yのウエイトにw_tだけ加える。なお、遷移しやすくなっている場合のウエイトw_tは 1 より大きく、遷移しにくくなっている場合のウエイトw_tは 1 より小さい。

- 状態xから状態yへの遷移時の状態xのウエイト
  摂動によって滞在しにくく[滞在しやすく]なっているため、その時刻での状態xのウエイトからw_tだけ減らす。

- 状態xから状態yへの遷移時の状態x, y以外のウエイト
  摂動による存在確率の変化がないため、ウエイトの変化は発生しない。

- 状態xでの滞在時のウエイト
  遷移していないため、ウエイトの変化は発生しない。この理由は以下の通りである。相関サンプリング法では、摂動体系におけるシナリオは基準体系と同一であると仮定している。状態xでの滞在時にウエイトが変化するということは、他の状態のウエイトがその分変化することとなる。なぜならば、ある事故シナリオのある時間ステップに着目した場合、全ての状態のウエイトを足し合わせると1になるはずであるからである。他の状態のウエイトが変化するということは、摂動体系におけるシナリオが基準状態と同一であるとの仮定に反する。従って、状態xでの滞在時にウエイトは変化しない。

ここで、一つのシナリオを例として挙げ、その時のウエイトの積算方法を示す。状態は状態a, b, c, dがあるときを考え、初期時刻に状態a、時刻t_2に状態b、時刻t_4に状態a、時刻t_5に状態cになっており、時刻t_7で計算時刻となり終了するシナリオを考える。そのシナリオ時のウエイトの積算方法を表 2-5 で示す。

表 2-5　ウエイトの作用のさせ方の一例（状態a（初期時刻）→状態b（時刻t_2）→状態a（時刻t_4）→状態c（時刻t_5）、時刻t_7で終了）

	状態a	状態b	状態c	状態d
t_0	1	0	0	0
t_1	1	0	0	0
t_2	$1-w_{t_2}$	$0+w_{t_2}$	0	0
t_3	$1-w_{t_2}$	$0+w_{t_2}$	0	0
t_4	$1-w_{t_2}+w_{t_4}$	$0+w_{t_2}-w_{t_4}$	0	0
t_5	$1-w_{t_2}+w_{t_4}-w_{t_5}$	$0+w_{t_2}-w_{t_4}$	w_{t_5}	0
t_6	$1-w_{t_2}+w_{t_4}-w_{t_5}$	$0+w_{t_2}-w_{t_4}$	w_{t_5}	0
t_7	$1-w_{t_2}+w_{t_4}-w_{t_5}$	$0+w_{t_2}-w_{t_4}$	w_{t_5}	0

2.5　本章のまとめ

本節では、本章のまとめを述べる。

- 2.1 節では、本章で CMMC 法と相関サンプリング法を用いた PRA の感度解析の概要について説明することを述べ、本章の概要を示した。
- 2.2 節では、確率論的リスク評価の理論を説明した。確率論的リスク評価の手法として、本論文で取り上げられている CMMC 法のほかの手法としてイベントツリー法の説明を行った。また、リスク重要度指標についても説明し、シミュレーションによる感度解析との比較も行った。
- 2.3 節では、CMMC 法の理論を説明した。CMMC 法を用いた PRA 手法として、CMMC カップリング手法の概要を説明し、2.4 節の相関サンプリング法の適用による感度解析の説明につなげた。
- 2.4 節では、相関サンプリング法の理論を説明した。CMMC 法を利用した PRA の感度解析手法として、相関サンプリング法の手順を示した。まず、中性子輸送において適用されている相関サンプリング法を CMMC 法へと拡張する方法について説明した。その後、本研究で注目する事故解析を対象として、事故シナリオに固有のウエイト及びシステムの状態に対するウエイトwの導出を行った。なお、ウエイトwの適用法および導出は一般的に説明すること容易ではなかったため、事故シナリオの具体例を挙げることによって、なるべくわかりやすく説明することを試みた。

参考文献

[1] 澤田憲人, “マルチユニットリスク評価への連続マルコフ過程モンテカルロ法の適用,” 修士論文, 名古屋大学, (2021).

[2] 原電エンジニアリング株式会社, “原子力規制人材育成講座/名古屋大学　確率論的リスク評価演習,”(2021).

[3] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, *et al*, “Equation of State Calculations by Fast Computing Machines,” *J. Chem. Phys.*, **21**, pp.1087-1092, (1953).

[4] Kschwendt,H. and Rief,H. , “TIMOC-A General Purpose Monte Carlo Code for Stationary and Time Dependent Neutron Transport,” EUR4519e, (1970).

[5] Gelbard,E.M. and Rief,H. , *et al.* , “MARC-A Multigroup Monte Carlo Programme for the Calculation of Capture Probabilities,” WAPD-TM-273, (1962).

3　単純なシステムを対象とした感度解析

3.1　本章の概要

本章では、単純なシステムを対象として CMMC 法と相関サンプリング法を用いて実施した感度解析について述べる。本章で CMMC 法と相関サンプリング法を用いた感度解析の妥当性を確認し、現実的なシステムである使用済燃料プールの感度解析へとつなげる。

本章では、単純なシステムとして、一つの機器を持つシステムと二つの機器を持つシステムを例にあげて機器の累積故障確率の解析を行う。感度解析には、第 2 章で述べた CMMC 法と相関サンプリング法を用いる。CMMC 法と相関サンプリング法を用いた計算結果、CMMC 法のみを用いた計算結果、解析解との比較を行う。CMMC 法のみを用いた計算結果と比較することで、統計誤差の影響を確認し、解析解と比較することで精度を評価する。なお、解析に用いる機器の故障率は仮想的な値を用い、解析の対象とする時間は短時間を想定する。また、二つの機器を持つシステムにおいては、機器の故障率が互いの機器の状態に依存すると仮定して解析を行う。

本章の構成は以下の通りである。

- 3.1 節では、本章の概要を述べる。
- 3.2 節では、一つの機器を持つシステムの解析について述べる。解析手順と解析条件を示したのち、計算結果と解析解との比較を行い、一つの機器を持つシステムでの感度解析の妥当性を確認する。
- 3.3 節では、二つの機器を持つシステムの解析について述べる。3.2 節と同様、解析手順と解析条件を示したのち、計算結果と解析解との比較を行い、二つの機器を持つシステムでの感度解析の妥当性を確認する。
- 3.4 節では、本章のまとめを述べる。

3.2　一つの機器を持つシステムの解析

本節では、CMMC 法と相関サンプリング法を用いた一つの機器を持つシステムの感度解析を行う。システム内に機器が一つ存在し、故障率λ [1/s]で故障する。機器は故障から健全へ変化するような回復は考えないとした。この時、状態遷移図は図 3-1 のようになる。

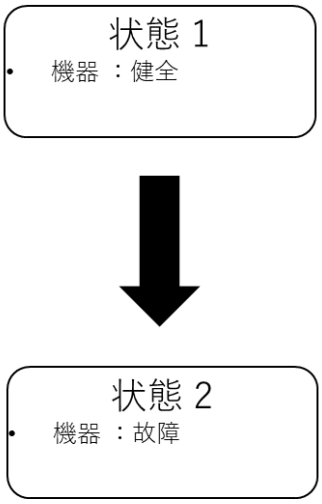


図 3-1　一つの機器を持つシステムの状態遷移図

3.2.1解析手順

この節では、一つの機器を持つシステムにおいての三つの解析方法について説明する。①CMMC 法と相関サンプリング法を用いた解析、および②CMMC 法のみを用いた解析については、それらの計算手順を示す。また、③理論式に基づく解析については、その解析解を導出する。

①CMMC 法と相関サンプリング法を用いた解析

CMMC 法と相関サンプリング法を用いたシステムの解析は次のような手順で行う。

1. 時刻$t = 0$において、機器を健全とする。
2. 各タイムステップで、タイムステップ幅に対応する故障確率と生成された乱数を比較することで、次のタイムステップの機器の状態を決定する。（乱数）＜（故障確率）のとき故障とする。不等式を満たさない場合は、前ステップでの状態を継続する。
3. 故障率、故障率の摂動割合、タイムステップ幅から、ウエイトを更新する。
4. 2.から 3.の手順を解析時間まで繰り返すことで事故シナリオと一連のウエイトを得る。
5. 1.から 5.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の事故シナリオとウエイトを得る。
6. 事故シナリオとウエイトから、基準状態と摂動状態の累積故障確率を求める。
7. 基準状態と摂動状態の累積故障確率から、累積故障確率の変化量を得る。

②CMMC 法のみを用いた解析

CMMC 法のみを用いたシステムの解析は次のような手順で行う。

1. 時刻$t = 0$において、機器を健全とする。
2. 各タイムステップで、タイムステップ幅に対応する故障確率と生成された乱数を比較

することで、次のタイムステップの機器の状態を決定する。（乱数）＜（故障確率）のとき故障とする。不等式を満たさない場合は、前ステップでの状態を継続する。

3. 2.の手順を解析時間まで繰り返すことで事故シナリオを得る。
4. 基準状態と摂動状態の故障率で 1.から 3.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の基準状態と摂動状態の事故シナリオを得る。
5. 基準状態と摂動状態の事故シナリオから、基準状態と摂動状態の累積故障確率を求める。
6. 基準状態と摂動状態の累積故障確率から、累積故障確率の変化量を得る。

③理論式に基づく解析

機器の累積故障確率Pは故障率λを用いて、式(3.1)に示すように解析的に与えられる。時間dtの間に増加する累積故障確率dPはその時刻tにおいて故障していない確率$(1-P)$と故障率λdtの積で書くことができる。

$$dP = (1-P)\lambda dt$$
$$\frac{dP}{dt} = (1-P)\lambda \tag{3.1}$$

式(3.1)を初期時刻$t=0$から時刻tまで積分すると、式(3.2)の解析解が得られる。

$$\int_0^{P(t)} \frac{1}{1-P} dP = \int_0^t \lambda \, dt$$
$$-\ln\big(1-P(t)\big) = \lambda t \tag{3.2}$$
$$P(t) = 1 - \exp(-\lambda t)$$

3.2.2解析条件

解析条件として、機器の故障率、CMMC 法の計算条件を述べる。

- 機器の故障率

本解析では、一つの機器を考慮する。機器に関する解析条件を表 3-1 に示す。

表 3-1　一つの機器を持つシステムの機器の解析条件

パラメータ	値	設定理由
故障率λ [1/s]	0.01	[1]
故障率の摂動割合r [-]	0.01	仮想的な値。故障率を 1%変化させることに相当

- CMMC 法の計算条件

CMMC 法の計算条件を表 3-2 に示す。CMMC 法のみを用いた計算と、CMMC 法と

相関サンプリング法を用いた計算を行うが、全て同じサンプル数で計算を行う。

表 3-2　一つの機器を持つシステムの CMMC 法の計算条件

パラメータ	値
解析対象時間 [s]	100
時間幅Δt [s]	0.1
サンプル数 [-]	10^5

3.2.3解析結果

基準状態、摂動状態および変化量(摂動-基準)の累積故障確率についての解析結果をそれぞれ図 3-2、図 3-3、図 3-4 に示す。図 3-2、図 3-3 の横軸は時間 [s] 、縦軸は累積故障確率 [-]を示している。図 3-4 の横軸は時間 [s]、縦軸は累積故障確率の変化量 [-]を示している。

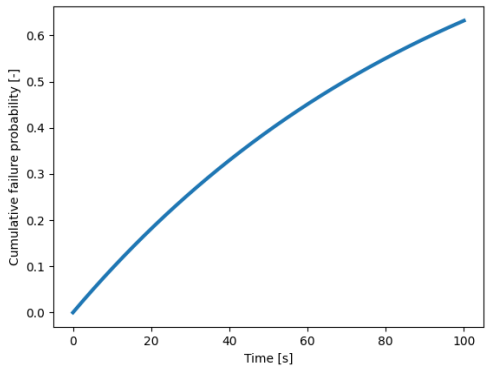


図 3-2　一つの機器を持つシステムの解析結果(基準状態)

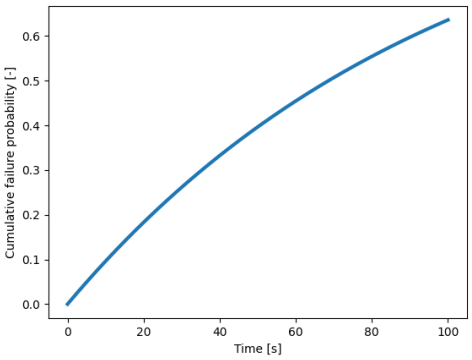
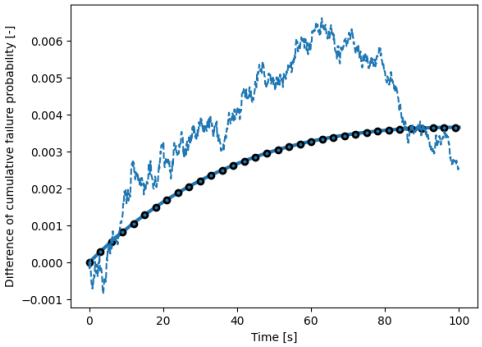


図 3-3　一つの機器を持つシステムの解析結果(摂動状態)



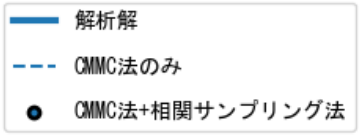


図 3-4　一つの機器を持つシステムの解析結果(変化量)

計算結果から、CMMC 法のみの計算結果は統計誤差のため解析解からずれているのに対し、CMMC 法と相関サンプリング法の計算結果は解析解と一致している。CMMC 法のみを用いた計算結果は、基準状態の累積故障確率計算値に付随する統計誤差と摂動状態の累積故障確率計算値に付随する統計誤差の二つの誤差が影響するが、CMMC 法と相関サンプリング法の計算結果は基準状態と摂動状態の変化量を基準状態の計算結果とウエイトを用いて解析的に計算しているため、解析解との差異が小さくなったと考える。以上のことから、一つの機器を持つシステムにおいて、相関サンプリング法による感度解析の妥当性を確認することが出来た。

3.3　二つの機器を持つシステムの解析

本節では、CMMC 法と相関サンプリング法を用いた二つの機器を持つシステムの感度解析を行う。システム内に機器 A と機器 B の二つ存在し、それぞれ故障率λ_A, λ_B [1/s]で故障する。機器は故障から健全へ変化するような回復は考えないとした。この時、状態遷移図は

図 3-1 のようになる。

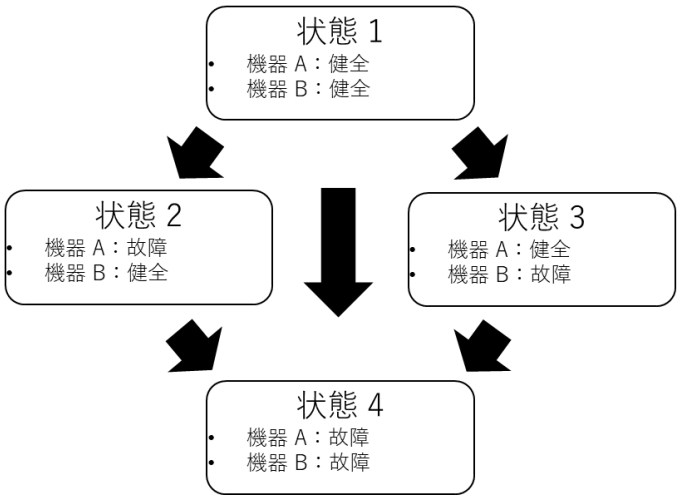


図 3-5　二つの機器を持つシステムの状態遷移図

3.3.1解析手順

この節では、二つの機器を持つシステムにおいての三つの解析方法について説明する。①CMMC 法と相関サンプリング法を用いた解析、②CMMC 法のみを用いた解析、および③解析的に求める方法についての計算手順を、それぞれ以下で示す。

①CMMC 法と相関サンプリング法を用いた解析

CMMC 法と相関サンプリング法を用いたシステムの状態の解析は次のような手順で行う。

1. 時刻$t = 0$において、二つの機器を健全とする。
2. お互いの機器の状態に応じて故障率を変化させる。
3. 各タイムステップで、タイムステップ幅分に対応する故障確率と生成された乱数を比較することで、次のタイムステップの機器の状態を決定する。（乱数）＜（故障確率）のとき故障とする。不等式を満たさない場合は、前ステップでの状態を継続する。
4. 故障率、故障率の摂動割合、タイムステップ幅から、ウエイトを更新する。
5. 2.から 4.の手順を解析時間まで繰り返すことで事故シナリオと一連のウエイトを得る。
6. 1.から 5.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の事故シナリオとウエイトを得る。
7. 事故シナリオとウエイトから、基準状態と摂動状態の存在確率と機器の累積故障確率を求める。
8. 基準状態と摂動状態の存在確率と累積故障確率の差異から、存在確率と機器の累積故障確率の変化量を得る。

②CMMC 法のみを用いた解析

CMMC 法のみを用いたシステムの解析は次のような手順で行う。

1. 時刻$t = 0$において、二つの機器を健全とする。
2. お互いの機器の状態に応じて故障率を変化させる。
3. 各タイムステップで、タイムステップ幅分に対応する故障確率と生成された乱数を比較することで、次のタイムステップの機器の状態を決定する。（乱数）<（故障確率）のとき故障とする。不等式を満たさない場合は、前ステップでの状態を継続する。
4. 2.から 3.の手順を解析時間まで繰り返すことで事故シナリオを得る。
5. 基準状態と摂動状態の故障率で 1.から 4.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の基準状態と摂動状態の事故シナリオを得る。
6. 基準状態と摂動状態の事故シナリオから、基準状態と摂動状態の存在確率と機器の累積故障確率を求める。
7. 基準状態と摂動状態の存在確率と累積故障確率の差異から、存在確率と機器の累積故障確率の変化量を得る。

③解析的に求める方法

解析的にシステムの状態を求める方法は次のような手順で行う。

1. システムの状態 1~4 の存在確率を$E_j(t)$とする。ただし、j: 1~4とする。初期時刻$t = 0$での存在確率は$E_1(0) = 1.0$、$E_2(0) = E_3(0) = E_4(0) = 0.0$となる。
2. 各タイムステップにおける状態jから状態kへの遷移確率$p_{j\to k}$を算出する。遷移確率については後で記述する。
3. 現在の存在確率と遷移確率を掛け合わせることで、次の時間ステップでの存在確率を得る。存在確率と遷移確率の掛け合わせについても後で記述する。
4. 機器Aの累積故障確率は$E_2(t) + E_4(t)$、機器Bの累積故障確率は$E_3(t) + E_4(t)$を計算することで求める。

タイムステップ Δt当たり機器 A が故障する確率は$\lambda_A \Delta t$、機器 B が故障する確率は$\lambda_B \Delta t$である。それぞれの状態からの遷移のベン図を図 3-6 に示す。

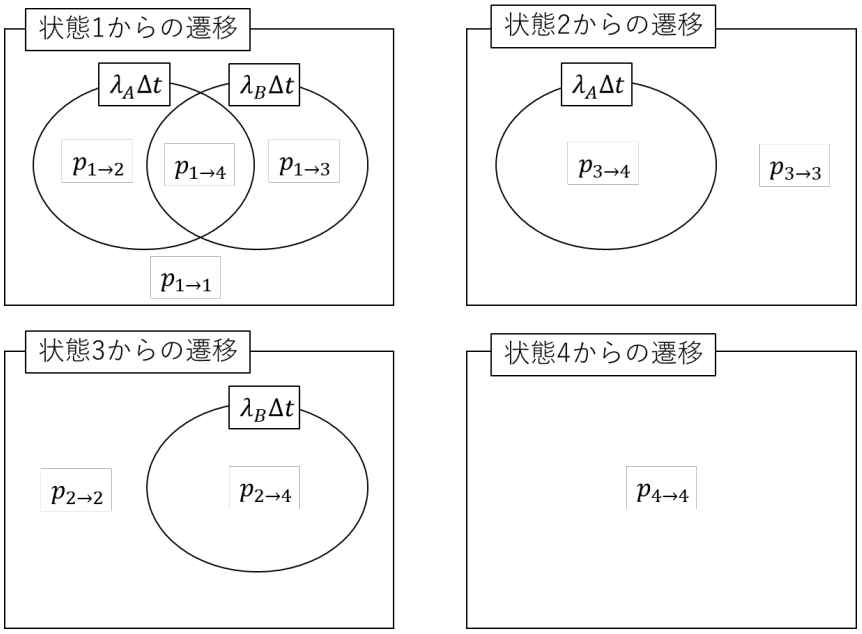


図 3-6　故障確率と遷移確率の関係

遷移確率は$p_{1\to2}, p_{1\to3}, p_{1\to4}, p_{2\to4}, p_{3\to4}$の 5 つあり、図 3-6 から式**(3.3)**のように計算できる。

$$\begin{aligned} p_{1\to2} &= \lambda_A\Delta t \times (1-\lambda_B\Delta t) \\ p_{1\to3} &= (1-\lambda_A\Delta t) \times \lambda_B\Delta t \\ p_{1\to4} &= \lambda_A\Delta t \times \lambda_B\Delta t \\ p_{2\to4} &= \lambda_B\Delta t \\ p_{3\to4} &= \lambda_A\Delta t \end{aligned} \tag{3.3}$$

また、遷移せずとどまる確率は$p_{1\to1}, p_{2\to2}, p_{3\to3}, p_{4\to4}$の 4 つあり、図 3-6 から式**(3.4)**のように計算できる。

$$\begin{aligned} p_{1\to1} &= (1-\lambda_A\Delta t) \times (1-\lambda_B\Delta t) \\ p_{2\to2} &= (1-\lambda_B\Delta t) \\ p_{3\to3} &= (1-\lambda_A\Delta t) \\ p_{4\to4} &= 1 \end{aligned} \tag{3.4}$$

求めた遷移確率を用いて、式**(3.5)**のように次の時間ステップでの存在確率を計算する。

$$\begin{aligned} E_1(\mathrm{t}_{i+1}) &= E_1(\mathrm{t}_i) \times p_{1\to1}(\mathrm{t}_i) \\ E_2(\mathrm{t}_{i+1}) &= E_1(\mathrm{t}_i) \times p_{1\to2}(\mathrm{t}_i) + E_2(\mathrm{t}_i) \times p_{2\to2}(\mathrm{t}_i) \\ E_3(\mathrm{t}_{i+1}) &= E_1(\mathrm{t}_i) \times p_{1\to3}(\mathrm{t}_i) + E_3(\mathrm{t}_i) \times p_{3\to3}(\mathrm{t}_i) \\ E_4(\mathrm{t}_{i+1}) &= E_1(\mathrm{t}_i) \times p_{1\to4}(\mathrm{t}_i) + E_2(\mathrm{t}_i) \times p_{2\to4}(\mathrm{t}_i) + E_3(\mathrm{t}_i) \times p_{3\to4}(\mathrm{t}_i) \\ &\quad + E_4(\mathrm{t}_i) \times p_{4\to4}(t_i) \end{aligned} \tag{3.5}$$

3.3.2解析条件

解析条件として、機器の故障率、CMMC 法の計算条件を述べる。

システムの故障率(単位時間当たり)は式(3.6)に従うと仮定した。

$$\lambda_X(t) = c_{1X}(1 - \exp(-c_2 t)) \tag{3.6}$$

ここで、各記号の定義は以下の通りである。

$\lambda_X(t)$：時刻tにおける単位時間当たりの要素Xの故障率［1/s］

c_{1X}：要素Xの状態定数［1/s］

c_2：故障時定数［1/s］

- 機器の故障率

本解析では、二つの機器を考慮する。機器の故障率は互いの機器の状態に依存すると仮定した。これは、使用済燃料プールなどの現実的なシステムの解析を行うためには、相互依存性を考慮する必要があるためである。機器Aが故障すると機器Bも故障しにくくなり、機器Bが故障すると機器Aが故障しやすいと仮定した。機器に関する解析条件を表 3-3 に示す。

表 3-3　二つの機器を持つシステムの機器の解析条件

パラメータ	値	設定理由
c_{1A}（B：健全/故障）［1/s］	0.2/0.8	[1]
c_{1B}（A：健全/故障）［1/s］	0.1/0.01	[1]
c_2［1/s］	0.001	[1]
A の故障率の摂動割合 r_{1A}（B：健全/故障）［1/s］	0.01/-0.02	仮想的な値
B の故障率の摂動割合 r_{1B}（A：健全/故障）［1/s］	0.03/0.04	仮想的な値

- CMMC 法の計算条件

CMMC 法の計算条件を表 3-4 に示す。一つの機器を持つシステムでの解析同様、CMMC 法のみを用いた計算と、CMMC 法と相関サンプリング法を用いた計算を行うが、全て同じサンプル数で計算を行う。

表 3-4　二つの機器を持つシステムの CMMC 法の計算条件

パラメータ	値
解析対象時間 [s]	100
時間幅Δt [s]	0.1
サンプル数 [-]	10^5

3.3.3解析結果

基準状態、摂動状態および変化量(摂動-基準)のシステムの状態の存在確率についての解析結果を図 3-7～図 3-9 に示す。基準状態、摂動状態および変化量(摂動-基準)の機器の累積故障確率についての解析結果を図 3-10～図 3-12 に示す。図 3-7、図 3-8 の横軸は時間 [s] 、縦軸は累積故障確率 [-]を示している。図 3-9 の横軸は時間 [s]、縦軸は累積故障確率の変化量 [-]を示している。また、図 3-10、図 3-11 の横軸は時間 [s]、縦軸は存在確率 [-]を示している。図 3-12 の横軸は時間 [s]、縦軸は存在確率の変化量 [-]を示している。

図 3-9、図 3-12 から、一つの機器を持つシステムと同様、CMMC 法のみの計算結果は統計誤差のため解析解からずれているのに対し、CMMC 法と相関サンプリング法の計算結果は解析解と一致している。その理由も 3.2.3 で一つの機器を持つシステムと同じように、統計誤差の伝播からきていると考える。以上のことから、故障率に相互依存性がある二つの機器を持つシステムに対しても相関サンプリング法による感度解析の妥当性を確認した。

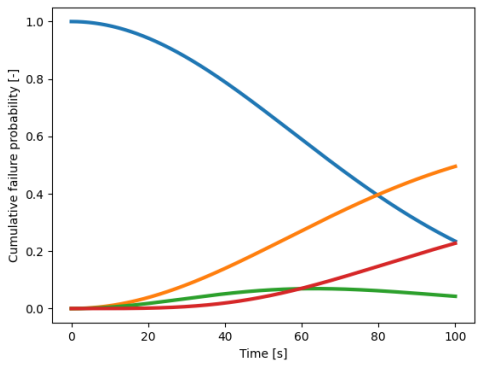


図 3-7　二つの機器を持つシステムの解析結果(基準状態でのシステムの状態)

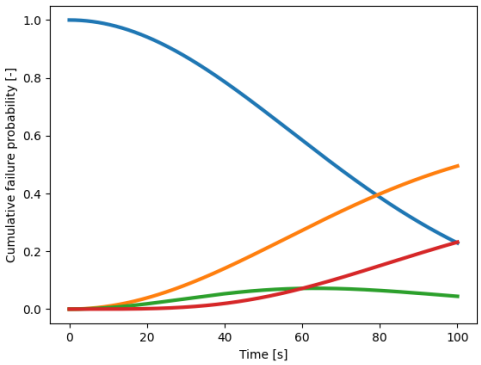


図 3-8　二つの機器を持つシステムの解析結果(摂動状態でのシステムの状態)

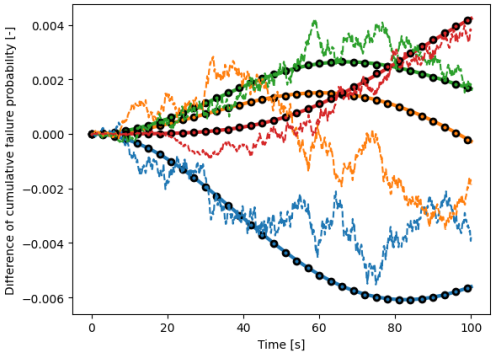


図 3-9　二つの機器を持つシステムの解析結果（システムの状態の変化量）

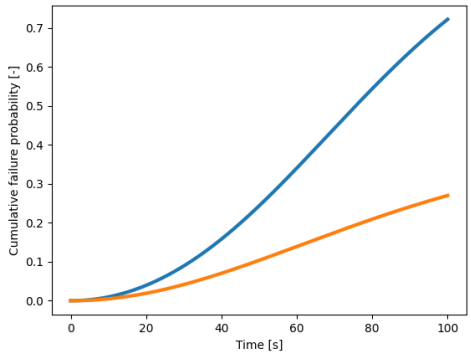


図 3-10　二つの機器を持つシステムの解析結果（基準状態での機器の状態）

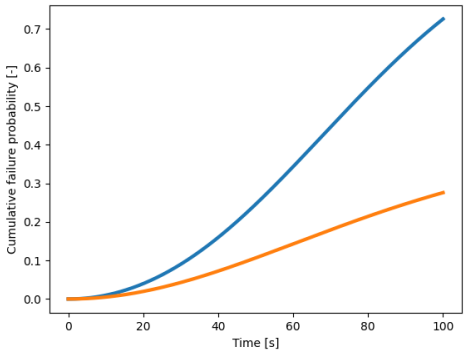


図 3-11　二つの機器を持つシステムの解析結果（摂動状態での機器の状態）

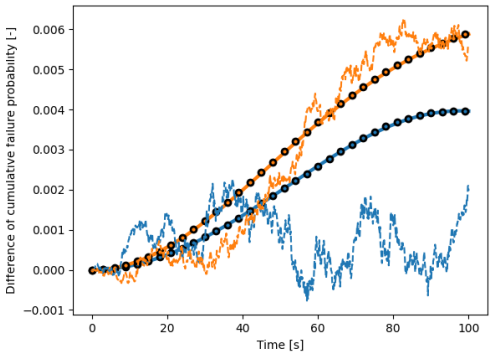


図 3-12　二つの機器を持つシステムの解析結果（機器の状態の変化量）

3.4　本章のまとめ

本節では、本章のまとめを述べる。

- 3.1 節では、単純なシステムを対象とした感度解析について説明することを述べ、本章の概要を示した。
- 3.2 節では、一つの機器を持つシステムの解析を行った。解析結果から、CMMC 法のみの計算では統計誤差のため解析解と差異が生じることを確認した。一方、CMMC 法と相関サンプリング法の計算結果が解析解と一致することを確認し、自作した相関サンプリング法の妥当性を確認できた。
- 3.3 節では、二つの機器を持つシステムの解析を行った。解析結果から、一つの機器を持つシステムと同様、CMMC 法のみの計算では統計誤差のため解析解と差異が生じることを確認し、一方、CMMC 法と相関サンプリング法の計算結果が解析解と一致することを確認した。また、故障率の相互依存性がある場合であっても、相関サンプリング法の妥当性を確認できた。

参考文献

[1] 澤田憲人, “マルチユニットリスク評価への連続マルコフ過程モンテカルロ法の適用,” 修士論文, 名古屋大学, (2021).

4　使用済燃料プールを対象とした感度解析

4.1　本章の概要

本章では、CMMC 法と相関サンプリング法を用いて実施した使用済燃料プール（Spent Fuel Pool：SFP）を対象としたリスク評価における感度解析について述べる。本章で実施する解析を通して、複数機器間の依存性を考慮した解析に対する CMMC 法と相関サンプリング法による感度解析の適用性の確認を行う。

では、まず沸騰水型軽水炉（BWR）プラントに設置されている SFP の簡易モデルを作成する。モデルは名古屋大学の澤田氏の修士論文[1]において作成されていたモデルをもとに作成した。作成した簡易モデルを用い、最終ヒートシンク喪失（Loss of Ultimate Heat Sink：LUHS）発生時の解析を行う。解析対象とする SFP の数（一基、二基）や考慮する緩和系の機器の数（消火系ポンプのみの一機、消火系ポンプと消防車の二機）を変えた複数の条件について解析を実施することで、マルチユニット解析で想定される多岐にわたる状況に対する CMMC 法と相関サンプリング法による感度解析の適用性を検証する。なお、緩和系とは事故進展を緩和するために実施されるアクシデントマネジメントを指す。本章では、比較的短時間で解析可能な仮想的なパラメータを用いた解析を実施する。

本章の構成は以下の通りである。

- 4.1 節では、本章の概要を述べる。
- 4.2 節では、本解析の条件を述べる。解析で想定した事故シーケンスと作成した SFP の解析モデル、及び解析モデルの入力パラメータについて説明する。
- 4.3 節では、単一の使用済燃料プールを対象とした解析について述べる。単一の使用済燃料プールの解析を行い、SFP 燃料損傷頻度（SFD）の感度解析の妥当性を確認することで、4.4 節での解析につなげる。
- 4.4 節では、複数の使用済燃料プールを対象とした解析について述べる。使用済燃料プール間において依存性を考慮した解析においての SFP 燃料損傷頻度（SFD）の感度解析の妥当性を確認する。
- 4.5 節では、本章のまとめを述べる。

4.2　解析モデルと解析条件

4.2.1想定する事故シーケンス

本項では、本章の解析で想定する事故シーケンス（SFP における LUHS）について述べる。

SFP の概要と原子力発電所における LUHS について説明する。SFP は原子炉に装荷され発電に使用した燃料（使用済燃料）、継続して使用する燃料（燃焼燃料）と未使用の燃料（新燃料）が保管されている。SFP は約 12 [m]の深さの水で満たされており、崩壊熱を発生させる燃料の冷却する役割を果たしている。通常時は、燃料プール冷却浄化系（Fuel Pool

Cooling and Filtering System : FPS 系）や残留熱除去系（Residual Heat Removal System : RHR 系）によってプール水を循環し、崩壊熱を除去する。これらプール水を循環し、崩壊熱を除去する機器が津波による機器の浸水、故障、停電などによって機能喪失すると、使用済燃料から発生する崩壊熱が除去できずプール水が高温になり蒸発による水位の低下、燃料の露出、燃料の温度上昇、燃料損傷の順に事故が進展する。FPS ポンプや RHR ポンプなどの崩壊熱を除去する機器の機能を失うことを最終ヒートシンク喪失（LUHS）と呼ぶ。この事故進展を緩和させるアクシデントマネジメント（Accident Management : AM）策の一つとして、消火系ポンプや消防車を用いて注水を行う。

本章では、FPS ポンプや RHR ポンプが何らかの起因事象により機能が喪失し、消火系ポンプや消防車によって燃料を冷却しなければならなくなった状態を初期状態として解析を行った。なお、4.3 節や 4.4 節では、対象とする SFP の個数や緩和系の機器の個数を変えた複数の条件で解析を行うことで、想定される様々な状況を模擬した。

4.2.2使用済燃料プールの解析モデル

本項では、本章の解析を行うために作成した SFP のモデルについて述べる。

SFP の事故進展は、既存の事故進展解析コードを用いることで解析できる。しかし、本研究では CMMC 法と相関サンプリング法による SFP を対象とした感度解析が目的であるため、長い計算時間を必要とする既存の事故進展解析コードではなく、物理現象や SFP の構成要素を簡略化した簡易解析モデルを作成し解析に用いた。モデルは名古屋大学の澤田氏の修士論文[1]において作成されていたモデルをもとに作成した。簡易解析モデルで想定した体系や物理現象について説明する。

簡易解析モデルの略図を図 4-1 に示す。本解析モデルでは、SFP を直方体状として取り扱う。考慮した内部構造物を表 4-1 に示す。簡略化したため内部構造物は表 4-1 に示すように一般的な主成分のみで構成されているとし、それぞれの配置を厳密には考慮していない。

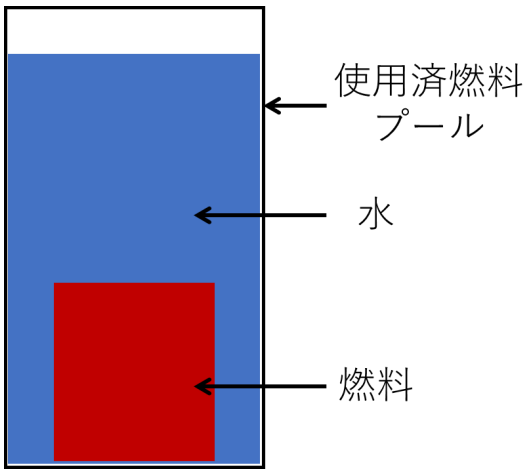


図 4-1　簡易解析モデルの略図（SFP）

表 4-1　考慮した内部構造物

構造物の名称	構造物の組成
燃料	UO_2
被覆管	Zr
燃料缶(チャンネルボックス)	Zr
燃料支持具上部	Zr及びステンレス鋼
燃料支持具下部	Zr

本解析モデルでは、SFP 内での発生熱と消費熱について考え、外部との熱のやり取りがない断熱状態を仮定した。そのため、タイムステップごとにプール内で発生する熱量と、SFP 内の構造物の顕熱およびプール水の顕熱および潜熱の総和が等しいとする。また、簡易化のため、プール水および内部構造物が一様に温度上昇すると仮定した。さらに、実際の事故ではプール水の蒸発が進み、SFP 水位が燃料の上端（Top of Active Fuel：TAF）に到達した後は燃料が自身の崩壊熱により損傷すると考えられるが、本解析モデルでは燃料損傷による影響を考慮せず、TAF 到達後も単純に水位が低下するものとして扱う。加えて、SFP 水位が燃料の下端（Bottom of Active Fuel：BAF）に到達後の燃料の温度上昇などの計算はしておらず、水の沸点である約 100℃が SFP 内の構造物の温度の最大値となっている。なお、本解析モデルでは水位が TAF に到達した時点で燃料損傷とする。

また、緩和系による SFP への注水の際、水位制御を行う。水位制御とは、SFP 水位がある一定の水位を超えることを防ぐ目的で、SFP 水位が基準水位に達したとき注水を停止する操作である。

本論文で作成した簡易解析モデルの計算結果が、アメリカの EPRI が所有する MAAP5[2]による詳細計算の結果とほぼ一致していることを確認している。

- SFP 内での発生熱

SFP 内での発生する熱として、SFP 内の使用済燃料及び燃焼燃料から発生する崩壊熱を考慮する。崩壊熱は時間経過とともに減衰するが、原子炉停止から長時間経過後を想定する場合、事故進展の時間スケールでは一定とみなすことができる。本解析では、原子炉停止から長時間経過後を想定し、崩壊熱は時間によらず一定とした。タイムステップあたりに発生する崩壊熱を以下の式により計算する。

$$\Delta U_{decay} = \sum_{j}^{N} W_j \, \Delta t \tag{4.1}$$

ここで、

ΔU_{decay}：タイムステップあたりに発生する崩壊熱 [kJ]

N：燃料の総数 [-]

W_j：燃料jの崩壊熱 [W]

Δt：タイムステップ [s]

- SFP 内での消費熱

以下で、消費熱について説明する。

- プール水や構造物の顕熱

1） 解析の前に、以下の式によりプール内の燃料、構造物およびプール水が水の沸点（100℃）まで温度上昇するのに必要な顕熱をあらかじめ計算する。

$$U_0 = \sum_{k}^{M} M_k S_k (T_{boiling} - T_0) \tag{4.2}$$

ここで、

U_0：考慮する物質の温度が水の沸点まで上昇するのに必要な熱量 [kJ]

M：考慮する物質の総種数 [-]

M_j：物質jの重量 [kg]

S_j：物質jの比熱 [kJ/kg/K]

$T_{boiling}$：水の沸点 [K]

T_0：プールの初期温度 [K]

2） 緩和系による注入した水によって、プール内の構造物およびプール水が水の沸点（100℃）まで温度上昇するのに必要な顕熱が増加する。タイムステップあたりに増加する顕熱を以下の式により計算する。なお、緩和系により注入される水の温度は、プールの初期温度と同一とする。

$$\Delta U_{sensible} = \Delta M_{water} S_{water} (T_{boiling} - T_0) \tag{4.3}$$

ここで、

$\Delta U_{sensible}$：タイムステップあたりに増加する顕熱

ΔM_{water}：タイムステップあたりに注入される水の質量 [kg]

S_{water}：水の比熱 [kJ/kg/K]

$T_{boiling}$：水の沸点 [K]

T_0：プールの初期温度 [K]

3) 以上のことを踏まえ、タイムステップi＋1におけるプール内の構造物およびプール水が水の沸点（100℃）まで温度上昇するために必要な顕熱は以下の式で計算する。

$$U_{i+1} = \begin{cases} U_i - \Delta U_{decay} + \Delta U_{sensible}, & U_i - \Delta U_{decay} + \Delta U_{sensible} > 0 \\ 0, & U_i - \Delta U_{decay} + \Delta U_{sensible} \leq 0 \end{cases} \quad (4.4)$$

ここで、

U_i：タイムステップiにおいて考慮する物質が水の沸点（100℃）まで温度上昇するために必要な熱量

ΔU_{decay}：タイムステップあたりに発生する崩壊熱 [kJ]

$\Delta U_{sensible}$：タイムステップあたりに増加する顕熱 [kJ]

- プール水の潜熱

プール水の潜熱は、以下のように取り扱う。

タイムステップあたりのプール水の蒸発量は以下の式により計算する。

$$\Delta M_{vapor} = \begin{cases} \dfrac{\Delta U_{decay} - \Delta U_{sensible} - U_i}{L_{water}}, & \Delta U_{decay} - \Delta U_{sensible} - U_i > 0 \\ 0, & \Delta U_{decay} - \Delta U_{sensible} - U_i < 0 \end{cases} \quad (4.5)$$

ここで、

ΔM_{vapor}：タイムステップあたりのプール水の蒸発量 [kg]

ΔU_{decay}：タイムステップあたりに発生する崩壊熱 [kJ]

$\Delta U_{sensible}$：タイムステップあたりに増加する顕熱 [kJ]

U_i：タイムステップiにおいて考慮する物質が水の沸点（100℃）まで温度上昇するために必要な熱量

L_{water}：水の蒸発潜熱 [kJ/kg]

- 解析モデルのフロー

以上のまとめとして、本解析モデルの計算フローを図 4-2 に示す。図 4-2 において、iはタイムステップ数、U_iはタイムステップiにおいて考慮する物質が水の沸点まで温度上昇するために必要な熱量、Timeは解析時間である。注水量は水位と基準水位、機器の流量をもとに計算する。温度はU_iと考慮する物質の質量、比熱をもとに計算する、もしくは注水量と水位、前ステップでの温度をもとに計算する。本研究では、後者で計算した。

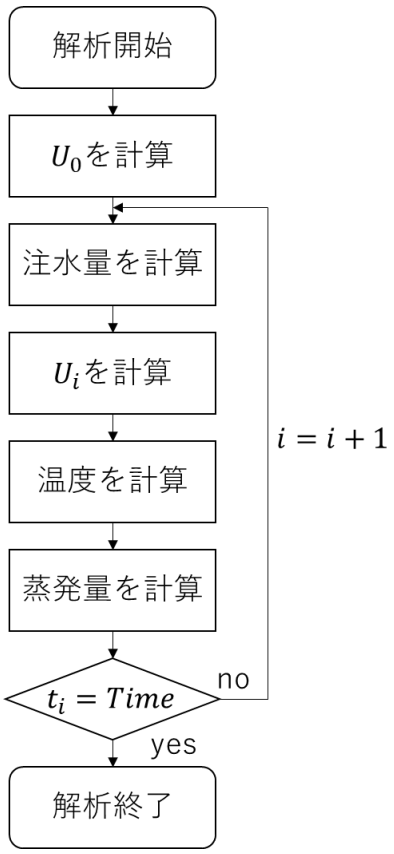


図 4-2　SFP 解析モデルのフロー

4.2.3入力パラメータ

本項では、作成した SFP モデルにおける入力パラメータについて述べる。

本解析モデルでは、アメリカの EPRI が所有する過酷事故進展解析コード MAAP5 のサンプルデータ（ピーチボトム原子力発電所の 2，3 号機（BWR プラント））の SFP に基づいた設定値としている [2]。この設定値を使用する場合、設定理由を「※」で記す。また、何らかの文献に基づく値を使用する場合、参考文献の番号を記載する。

表 4-2　SFP 形状に関する入力パラメータ

パラメータ	値	設定理由
床からの高さ [m]	22.58	※
床面積 [m^2]	131.3	※
プール内燃料体数 [体]	1020 (燃料 1：255、 燃料 2：765)	[1]
被覆管 Zr 重量 [kg]	37,332	[1]
燃料缶(チャンネルボックス)Zr 重量 [kg]	33,354	[1]
燃料支持具上部 Zr 重量 [kg]	760	※
燃料支持具上部ステンレス鋼重量 [kg]	400	※
有効燃料頂部（TAF） [m]	4.13	[1]
SFP 水位制御基準水位 [m]	12.9	[1]

表 4-3　解析の初期条件に関する入力パラメータ

パラメータ	値	設定理由
SFP 初期水温 [℃]	35	※
SFP 初期水位 [m]	11.9	※
燃料 1 の一体あたりの崩壊熱 [W]	7,400	[1]
燃料 2 の一体あたりの崩壊熱 [W]	1,700	[1]

表 4-4　各種の物性値

パラメータ	値	設定理由
水比熱 [kJ/kg/K]	4.2	[3]
水潜熱 [kJ/kg]	2,257	[3]
UO_2比熱 [kJ/kg/K]	0.373	[4]
Zr 比熱 [kJ/kg/K]	0.278	[5]
ステンレス鋼比熱 [kJ/kg/K]	0.46	[6]

4.3　単一の使用済燃料プールを対象とした解析

本節では、CMMC 法と相関サンプリング法を用いた単一 SFP に対する PRA の感度解析を行う。一つの SFP に対して、緩和系として消火系ポンプ 1 つと消防車 1 台を考える。LUHS が発生した初期時刻において消火系ポンプが作動した状態(健全)であるが、ある故障率に従って故障する。消火系ポンプが故障すると、ある起動率に従って消防車が起動する。ただし、起動した消防車は停止しないとする。このような事故シナリオを考える時、SFP に

おける状態遷移は図 4-3 のようになる。機器が 2 つあるため、状態は2^2通り存在するが、消火系ポンプと消防車が同時に機能していることはないので、あり得る状態は 3 つである。

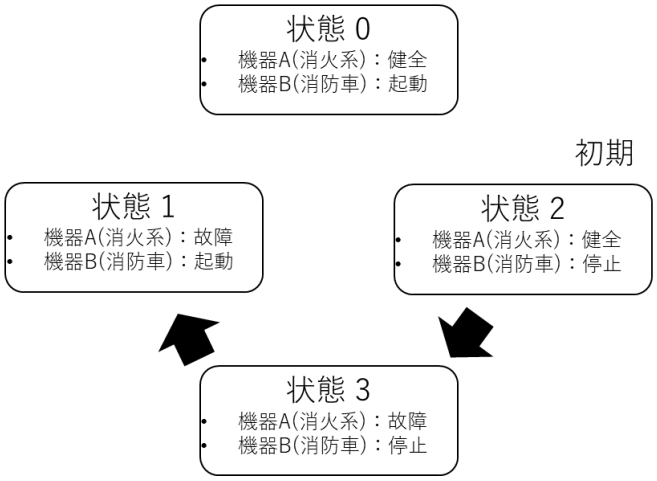


図 4-3　単一の使用済燃料プールの状態遷移図

4.3.1解析手順

CMMC 法と相関サンプリング法を用いたシステムの状態の解析は次のような手順で行う。なお、解析の初期状態として、使用済燃料プールの通常の冷却系は機能喪失しているとする。

1. 時刻$t = 0$において、緩和系の消火系ポンプを健全、消防車を停止中とする。
2. 消火系ポンプが健全の時、時間ステップ内における消火系ポンプの故障確率を用意する。消火系ポンプが故障かつ消防車が停止中の時、次の緩和系として消防車を考慮するため、消防車の起動確率を用意する。消火系ポンプが故障かつ消防車が健全の時、故障率や起動率を用意する必要はない。
3. 各タイムステップで、時間ステップ内の故障確率や起動確率と生成された乱数を比較することで、次のタイムステップの機器の状態を決定する。（乱数）<（故障確率）のとき故障、（乱数）<（起動確率）のとき起動とする。不等式を満たさない場合は、前ステップでの状態を継続する。
4. 故障率と起動率、それらの摂動割合、時間幅から、ウエイトを更新する。
5. 2.から 4.の手順を解析時間まで繰り返すことで事故シナリオと一連のウエイトを得る。
6. 1.から 5.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の事故シナリオとウエイトを得る。
7. 事故シナリオとウエイトから、基準状態と摂動状態における各状態の存在確率、各機器の累積故障確率、SFP 燃料損傷頻度、基準状態の温度と水位を求める。摂動状態の SFP 燃料損傷頻度の計算方法については後述する。
8. 基準状態と摂動状態における各状態の存在確率、累積故障確率と SFP 燃料損傷頻度か

ら、存在確率、累積故障確率と SFP 燃料損傷頻度の変化量を得る。

摂動状態における SFP 燃料損傷頻度は、基準状態における燃料損傷のシナリオに燃料損傷した時点t_iでの事故シナリオ固有のウエイトw_{t_i}を掛けることで摂動状態における燃料損傷のシナリオを得る。摂動状態における燃料損傷のシナリオの作成のイメージを図 4-4 に示す。作成した摂動状態における燃料損傷のシナリオの平均値を求めることで、摂動状態における SFP 燃料損傷頻度を得る。

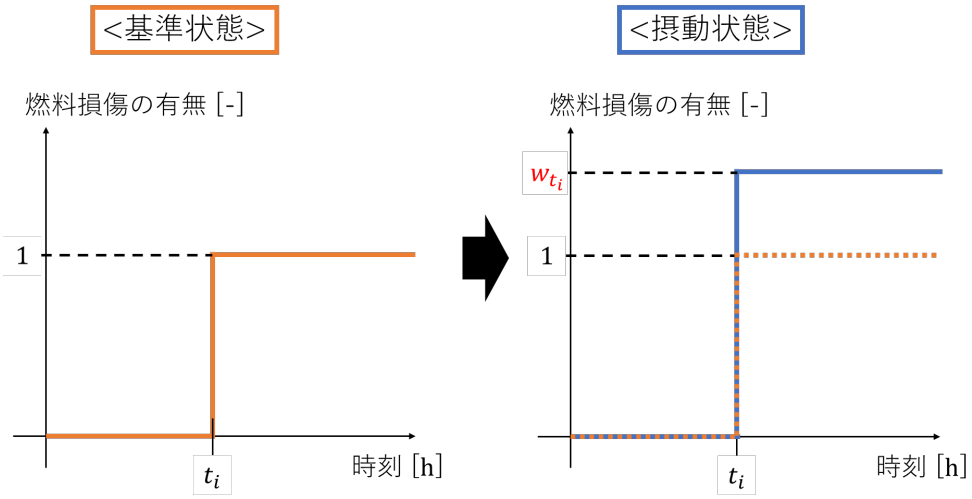


図 4-4　摂動状態における燃料損傷シナリオの作成

4.3.2解析条件

解析条件として、SFP モデルの条件、緩和系の条件、シミュレーションの条件を述べる。

- SFP モデルの条件

  モデルの入力パラメータは 4.2.3 項で示した値とした。解析では、単一の SFP を対象とした。

- 緩和系の条件

  本解析では、緩和系として消火系ポンプと消防車を考慮する。消火系ポンプと消防車に関する解析条件を表 4-5 に示す。ここで用いる消火系ポンプの故障率とその摂動割合、消防車の起動率とその摂動割合は相関サンプリング法の適用性を確認するための仮想的な値である。

表 4-5　単一 SFP の緩和系の解析条件

パラメータ	値	設定理由
消火系ポンプ流量［t/hr］	170	[1]
消防車流量［t/hr］	160	[1]
消火系ポンプ機能故障率［1/hr］	0.005	[1]
消防車起動率［1/hr］	0.005	[1]
消火系ポンプ機能故障率摂動割合［-］	0.01	仮想的な値
消防車起動率摂動割合［-］	0.01	仮想的な値

- シミュレーションの条件

  数値シミュレーション上の条件を表 4-6 に示す。

表 4-6　単一 SFP の数値シミュレーション上の条件

パラメータ	値
解析対象時間［hr］	1,000
タイムステップ幅［s］	300
サンプル数［-］	10^5

4.3.3解析結果

最初に、解析結果から 20 サンプル取り出した時の基準状態における温度と水位の解析結果を図 4-5、図 4-6 に示す。後に示す図 4-7、図 4-8 は CMMC 法で計算した 10 万サンプルを統計処理した結果となっており、平均値を示す形となっている。

解析結果を図 4-7～図 4-17 に示す。第 3 章では機器の故障を考えていたため、累積故障確率という言葉を使用していたが、消防車は故障ではなく停止しているため、累積停止確率という言葉に変更している。

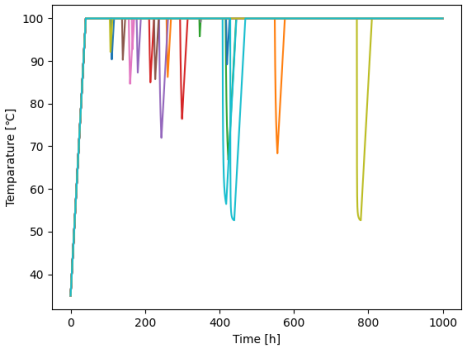


図 4-5　基準状態での各サンプルの温度

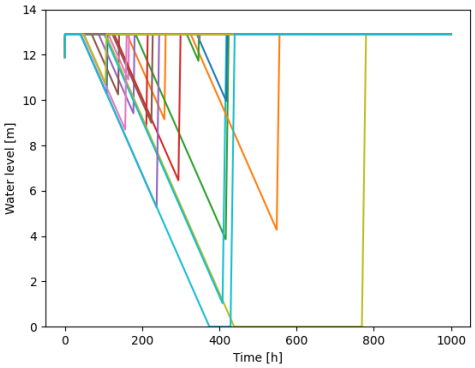


図 4-6　基準状態での各サンプルの水位

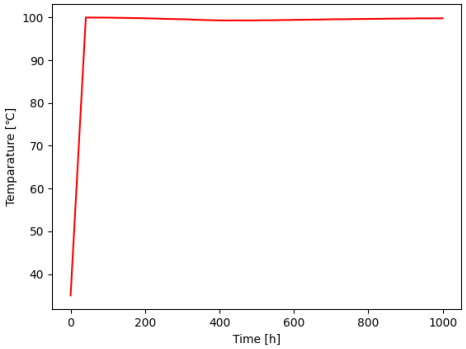


図 4-7　基準状態での温度
(全サンプル平均値)

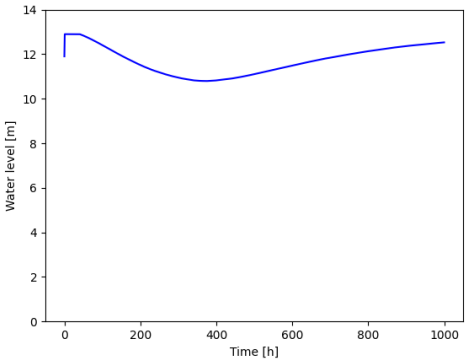


図 4-8　基準状態での水位
(全サンプル平均値)

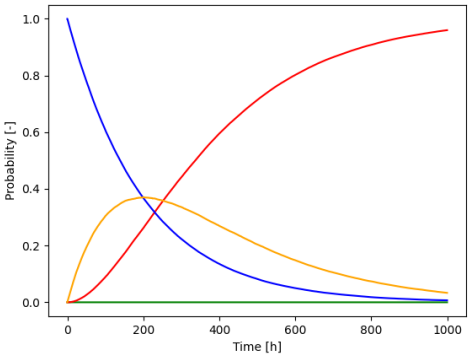


図 4-9　基準状態での各状態の存在確率

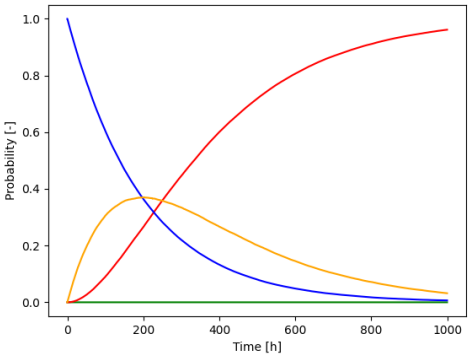
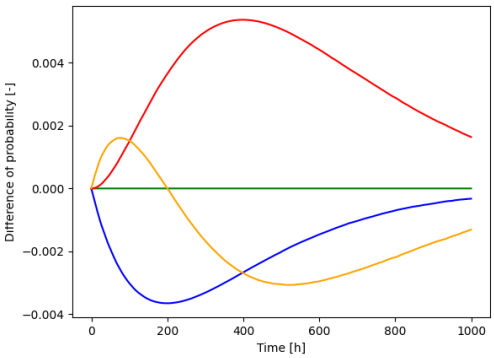


図 4-10　摂動状態での各状態の存在確率



存在確率の凡例

緑色：状態 0

赤色：状態 1

青色：状態 2

オレンジ色：状態 3

図 4-11　各状態の存在確率の変化量
(摂動状態-基準状態)

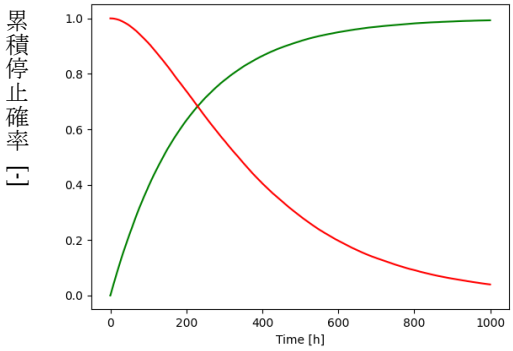


図 4-12　基準状態での各機器の累積停止確率

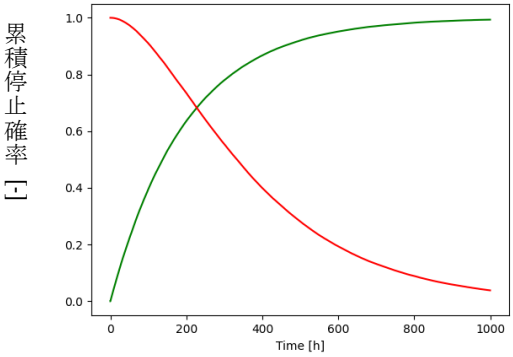


図 4-13　摂動状態での各機器の累積停止確率

累積停止確率の凡例

緑色：消火系ポンプ

赤色：消防車

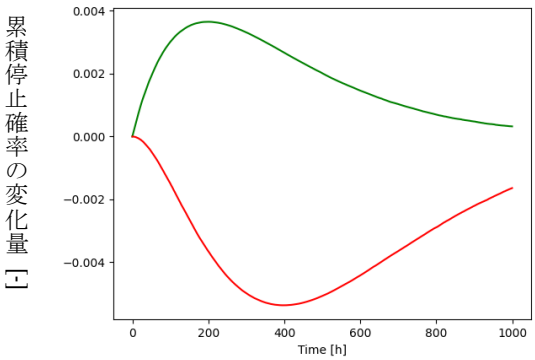


図 4-14　各機器の累積停止確率の変化量
(摂動状態-基準状態)

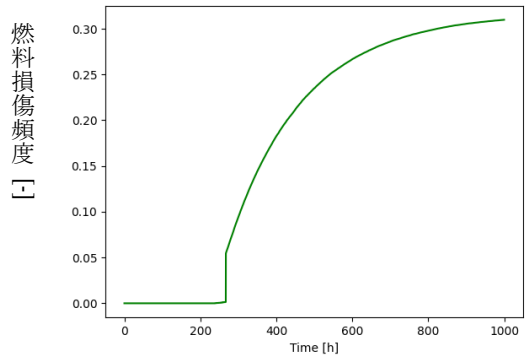


図 4-15　基準状態での SFP 燃料損傷頻度

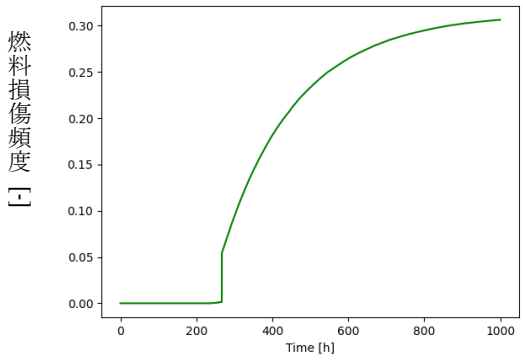


図 4-16　摂動状態での SFP 燃料損傷頻度

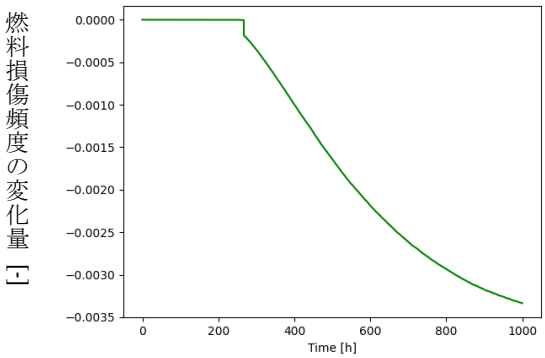


図 4-17　SFP 燃料損傷頻度の変化量(摂動状態-基準状態)

先行研究[1]の計算結果と今回の計算結果を比較し、水位の平均値や累積停止確率が正しく計算できていることを確認した。

図 4-7 について、温度は SFP に新たに水が入らない限り温度が下がらないため、平均値は温度が 100℃に達した後はほぼ 100℃を示すことになる。

図 4-8 について、水位はまず、消火系ポンプにより水位制御が起こるまで注水が行われるため一定値を保つ。その後消火系ポンプが故障するケースについては、消防車が起動するまで水位が下がり続ける。消防車が起動した後は、水位が上限まで上がり続ける。

図 4-9、図 4-10 は故障率を摂動する割合が小さいため(故障率を相対値で 0.01 だけ摂動)、両者はほぼ同じグラフとなる。状態 0 の存在確率は消火系ポンプと消防車が同時に作動しない仮定のため、常に 0 の値となる。状態 2(消火系健全、消防車停止)は初期状態であるため、存在確率 1 から始まり、時間とともに消火系ポンプが故障し減少する。状態 3(消化系故障、消防車停止)の存在確率は初期の存在確率 0 から始まるが状態 2 から消火系ポンプが故障するシナリオにより存在確率が増加した後、ある確率で消防車が起動して減少する。最後に状態 1(消火系故障、消防車起動)は初期の 0 から始まり、状態 3 において消防車が起動

するシーケンスが増加することで存在確率が増える。

図 4-11 について、状態 2(消火系健全、消防車停止)の存在確率は、摂動状態の方が小さくなる。これは、消火系の故障確率が摂動により増加しているためである。状態 1(消火系故障、消防車起動)については、摂動状態の存在確率が大きくなっているが、これは、摂動状態において消火系の故障がより多く発生し、状態 1→状態 3→状態 2 の遷移が発生する確率が大きくなっていることによる。状態 3 については、変化は比較的小さい。これは、消火系の故障確率増加と消防車の起動確率増加の効果が相殺しているためである。

図 4-12、図 4-13 も摂動割合が小さいため、基準状態と摂動状態はほぼ同じグラフとなる。消火系ポンプは故障のみが発生するため、累積停止確率は単調増加するグラフを示す。消防車は変化として起動のみが発生するため、累積停止確率は単調減少するグラフを示す。

図 4-14 について、消火系の累積停止確率は、摂動状態の方が大きくなる。これは、消火系の故障確率が摂動により増加しているためである。約 200 [hr]でピークをむかえ、その後、累積停止確率の変化量は減少する。減少する理由は、時間が進むにつれて、消火系が故障していない確率が減っていくためである。消防車については、摂動状態の累積停止確率が小さくなっている。これは、摂動状態において消火系の故障がより多く発生し、消防車の起動へ向かうシナリオが増えることと、消防車の起動確率が摂動により増加していることが起因している。

図 4-15、図 4-16 も摂動割合が小さいため、基準状態と摂動状態はほぼ同じグラフとなる。まず、初期状態から 200 [hr]程度までは燃料損傷頻度がほぼゼロとなっている。これは、消火系ポンプが早い段階で故障したとしても、SFP の水位が TAF に到達するには時間がかかることによる。SFP 内の温度が 100℃になった時刻において、消火系ポンプが故障しており、水位が TAF に至るまで消防車が起動しないシナリオが、約 270 [hr]で燃料損傷頻度が急激に増加することに寄与する。すなわち、初期温度から 100℃に至るまでの時間において消火系ポンプが故障しているすべてのシナリオは、100℃になった時刻において消火系ポンプが故障しているシナリオと数えられる。このシナリオが訳 270 [hr]で燃料損傷に至るため、燃料損傷頻度がこの時点で急激に増加することになる。その後、100℃に達した後に故障した消火系ポンプの影響で徐々に燃料損傷頻度が増加するが、計算時間末期になると、消防車が起動する影響で燃料損傷頻度が増加しにくくなる。

図 4-17 について、故障率と起動率の摂動の影響により約 270 [hr]以降で単調減少を示す。消火系ポンプと消防車がともに起動していない状態 3 の時間が長いと燃料損傷に至るため、消防車の起動率が増える摂動を与えると状態 3 の時間が短くなり燃料損傷に至りにくくなるため、燃料損傷頻度が負の値を示す。

3.3 節の二つの機器を持つシステムの解析に関する検討に基づくと、累積停止確率や存在確率の変化量も計算できていると考えている。また、水位が正しく計算できているため、基準状態の SFP 燃料損傷頻度も計算できていると考えている。

比較のため、図 4-17 の燃料損傷頻度の変化量の計算結果に対して、1 万サンプルと 100 万サンプルで計算を行った CMMC 法のみでの計算結果を追加した図を図 4-18 に示す。

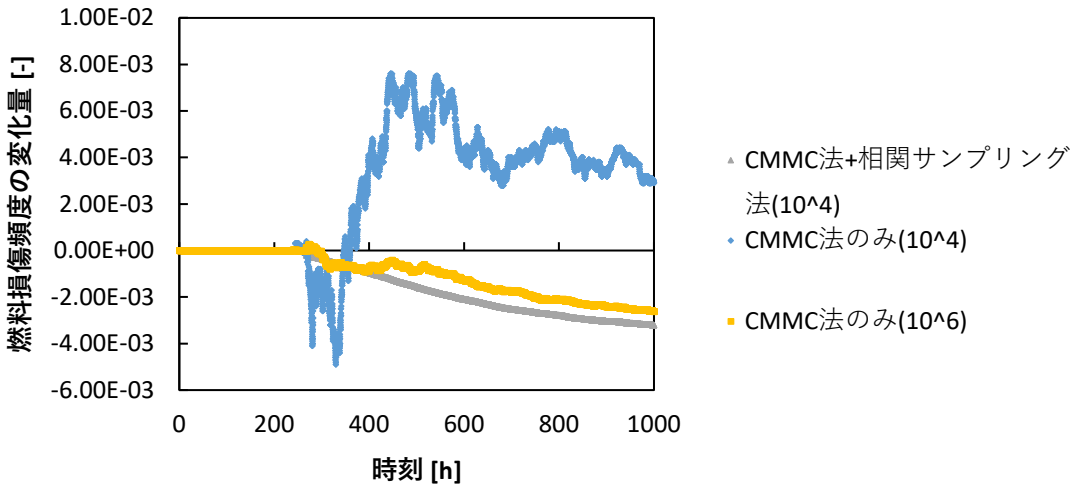


図 4-18　SFP の燃料損傷頻度の変化量(CMMC 法のみでの計算結果あり)

図 4-18 から、100 万サンプルでの CMMC 法のみの計算結果においても統計誤差の影響で振動していることがわかる。すなわち、CMMC 法のみで計算する場合にはより多くのサンプル数を用意する必要があるといえる。また、このことから、少ないサンプル数で計算を行うことが可能である相関サンプリング法の有効性が確認できる。

4.4　複数の使用済燃料プール間の依存性を考慮した解析

本節では、CMMC 法と相関サンプリング法を用いた複数の SFP の PRA の感度解析を行う。二つの SFP に対して、緩和系として消火系ポンプ 2 つと消防車 2 台を考える。一つの SFP に対して、消火系ポンプと消防車の一つずつ緩和系を用意する。初期時刻において消火系ポンプが作動した状態(健全)であるが、ある故障率に従って故障する。消化系ポンプが故障すると、ある起動率に従って同じ SFP に対して用意している消防車が起動する。ただし、起動後消防車は停止しないとする。このような事故シナリオを考える時、SFP における状態遷移は図 4-19 となる。機器が 4 つあるため、状態は2^4通り存在するが、消火系ポンプと消防車が同時に機能していることはないので、あり得る状態は 9 つである。

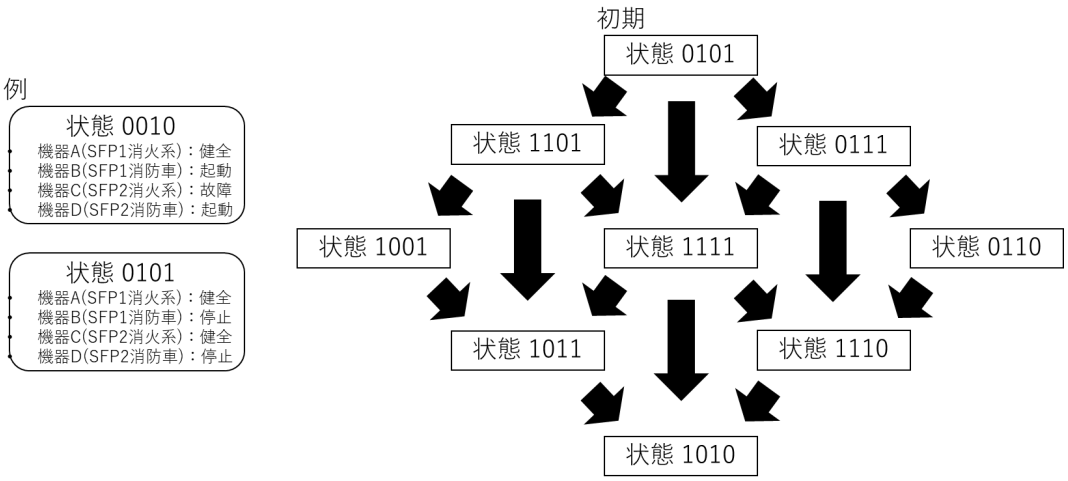


図 4-19　複数の使用済燃料プールの状態遷移図

4.4.1解析手順

CMMC 法と相関サンプリング法を用いたシステムの状態の解析は次のような手順で行う。なお、解析の初期状態として、使用済燃料プールの通常の冷却系は機能喪失しているとする。

1. 時刻$t = 0$において、緩和系の消火系ポンプ 1, 2 ともに健全、消防車 1, 2 ともに停止中とする。
2. 消火系ポンプが健全の時、時間ステップ内における消火系ポンプの故障確率を用意する。消火系ポンプが故障かつ消防車が停止中の時、消防車の起動率を用意する。消火系ポンプが故障かつ消防車が健全の時、故障率や起動率を用意する必要はない。一つの SFP の状態によって以上のことを考えるが、用意する故障率（もしくは起動率）は互いの SFP が燃料損傷に至っているかで変化させる。
3. 各タイムステップで、時間ステップ内の故障確率と生成された乱数を比較することで、次のタイムステップの機器の状態を決定する。（乱数）<（故障確率）のとき故障、（乱数）<（起動確率）のとき健全とする。不等式を満たさない場合は、前ステップでの状態を継続する。
4. 故障率と起動率、それらの摂動割合と時間幅から、相関サンプリング法のウエイトを更新する。SFP は 2 つあるが、一つのシステムとして捉えているため、用意するウエイトは一つで足りる。
5. 2.から 4.の手順を解析時間まで繰り返すことで事故シナリオと一連のウエイトを得る。
6. 1.から 5.の手順を設定したサンプル数だけ繰り返すことで、解析サンプル数分の事故シナリオとウエイトを得る。
7. 事故シナリオとウエイトから、基準状態と摂動状態における各状態の存在確率と、各機器の累積故障確率、SFP 燃料損傷頻度、基準状態の温度と水位を求める。
8. 基準状態と摂動状態における各状態の存在確率、累積故障確率と SFP 燃料損傷頻度か

ら、存在確率、累積故障確率と SFP 燃料損傷頻度の変化量を得る。

4.4.2解析条件

解析条件として、SFP モデルの条件、緩和系の条件、シミュレーションの条件を述べる。

- SFP モデルの条件

  モデルの入力パラメータは 4.2.3 項で示した値とした。解析では、二つの SFP を対象とした。

- 緩和系の条件

  本解析では、緩和系として消火系ポンプと消防車を考慮する。二つの SFP をそれぞれに消火系ポンプ 1 つ、消防車 1 台を緩和系として用意している。消防車による注水は運転員が建屋付近で消防車のホースを建屋側と接続する必要があるため、作業領域空間線量が高い場合に運転員の作業に制限がかかり、消防車による注水に失敗する可能性が高くなる。隣接する建屋内の SFP で燃料損傷が生じている場合に線量が上がり、消防車起動率が低下すると仮定した。システムの状態を図 4-20 に示す。また、消火系ポンプと消防車に関する解析条件を表 4-7 に示す。

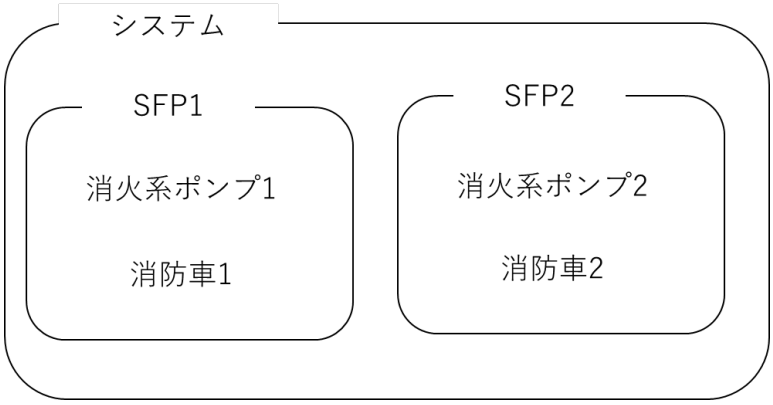


図 4-20　二つの SFP を持つシステムのベン図

表 4-7　複数の SFP の緩和系の解析条件

パラメータ	値	設定理由
消火系ポンプ 1, 2 流量 [t/hr]	170	[1]
消防車 1, 2 流量 [t/hr]	160	[1]
消火系ポンプ 1, 2 故障率 [1/hr]	0.005	[1]
消防車 1 起動率 [1/hr]	通常時：0.005 SFP2 燃料損傷時： 0.0005	[1]
消防車 2 起動率 [1/hr]	通常時：0.01 SFP1 燃料損傷時： 0.0001	[1]
消火系ポンプ 1, 2 故障率摂動割合 [-]	0.01	仮想的な値
消防車 1, 2 起動率摂動割合 [-]	0.01	仮想的な値

- シミュレーションの条件

  数値シミュレーション上の条件を表 4-8 に示す。

表 4-8　複数の SFP の数値シミュレーション上の条件

パラメータ	値
解析対象時間 [hr]	1,000
タイムステップ幅 [s]	300
サンプル数 [-]	10^4

4.4.3解析結果

解析結果を図 4-21～図 4-28 に示す。図 4-21 の水温、図 4-22 の水位に関しては、赤色が SFP1 の水温、温度を示しており、青色が SFP2 の水温、温度を示している。また、4.3.3 項同様、温度や水位は 10 万サンプルの平均値を示している。

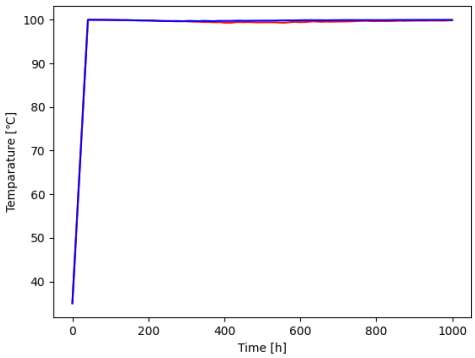


図 4-21　複数 SFP の温度
(全サンプル平均値)

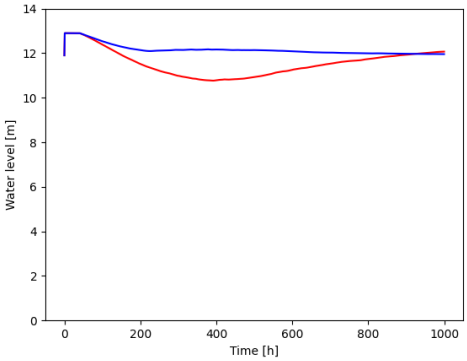


図 4-22　複数 SFP の水位
(全サンプル平均値)

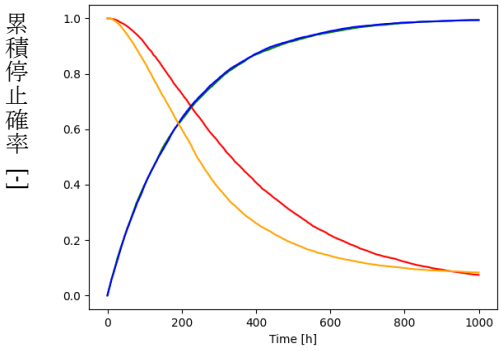


図 4-23　複数 SFP の基準の累積停止確率

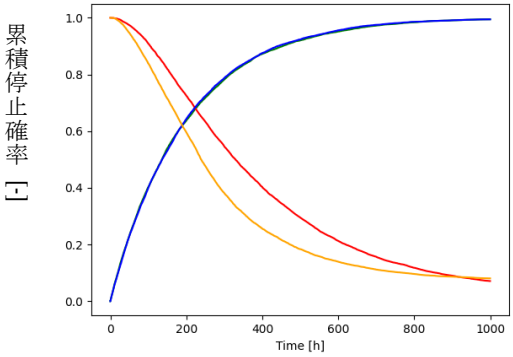


図 4-24　複数 SFP の揺動の累積停止確率

累積停止確率の凡例
緑色：消火系ポンプ 1
赤色：消防車 1
青色：消火系ポンプ 2
オレンジ色：消防車 2

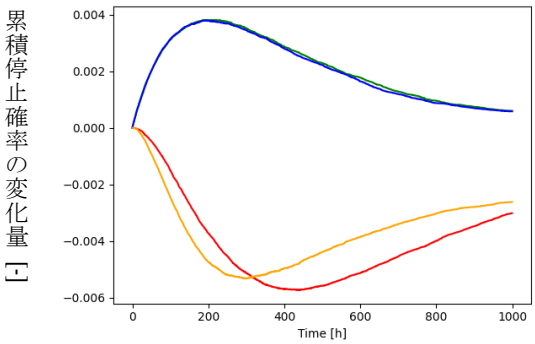


図 4-25　複数 SFP の累積停止確率の変化量

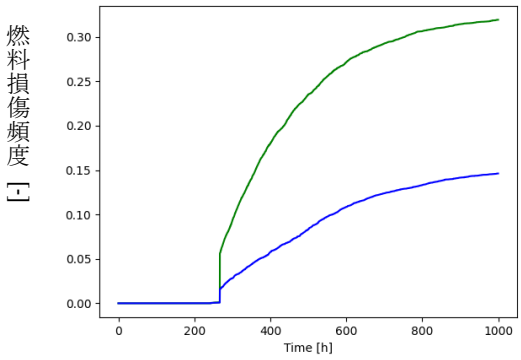


図 4-26　複数 SFP の基準の SFP 燃料損傷頻度

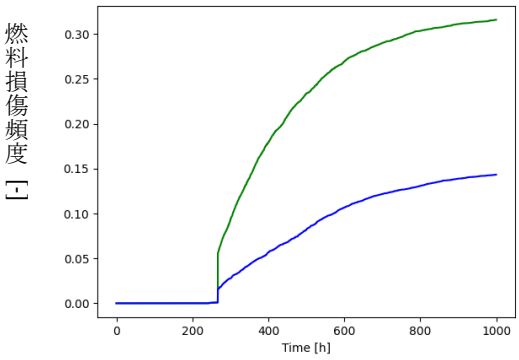


図 4-27　複数 SFP の摂動の SFP 燃料損傷頻度

SFP 燃料損傷頻度の凡例

緑色：SFP1

青色：SFP2

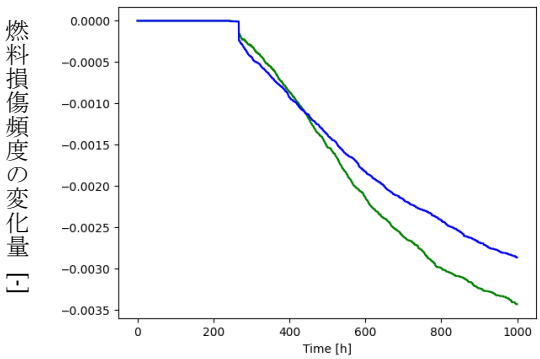


図 4-28　複数 SFP の SFP 燃料損傷頻度の変化量

先行研究の計算結果[1]と今回の計算結果を比較し、水位や累積停止確率が正しく計算できていることを確認した。単一 SFP と比較して、消防車の起動率が変化しているが、近い値を用いており、同じような計算結果が得られているため、パラメータごとの挙動の説明は割愛する。ここでは、SFP1 と SFP2 の計算結果の違いを記述する。消火系ポンプは SFP1、SFP2 どちらも同じ故障率としているため、異なる起動率を設定している消防車に着目する。

図 4-21 について、SFP1 の温度が SFP2 よりも少し低い値を示している。SFP1 における消防車の起動率が SFP2 のそれと比べて低いため、注水が遅れ蒸発する水量が増え、消防車による注水量が多くなったので、温度が低くなったと考える。

図 4-22 について、SFP1 の水位が SFP2 よりも少し低い値を示している。SFP1 における消防車の起動率が SFP2 のそれと比べて低いため、消防車による注水が遅れたからであると考える。

図 4-23、図 4-24 について、消防車 2 の方が消防車 1 よりも停止確率が低いため、消防車 2 が起動している確率がほとんどの時刻において消防車 1 より高いといえる。これは、

燃料損傷が起こっていないときの消防車 2 の起動率が消防車 1 の起動率より大きいためであると考える。末期において累積停止確率の大小が入れ替わるのは、燃料損傷に至った時の起動率については消防車 1 の方が高いためであると考える。

図 4-25 について、消防車 1 と消防車 2 の累積停止確率の変化のピークを比較すると、消防車 1 の方が消防車 2 と比べて遅い時間に表れる。これは、消防車 1 の起動率が小さいため、停止確率の変化が小さく、従って変化量においてもピークが遅れる形で表れたと考える。

図 4-26、図 4-27 について、表 4-7 で示したように燃料損傷が発生する前の起動率は消防車 1 の方が小さいため、消防車 1 による緩和の効果が小さくなり、結果として SFP1 の燃料損傷頻度の方が大きな値となったと考える。

図 4-28 について、燃料損傷が発生する以前は消防車 2 の起動率の方が大きいため、時刻 270 [hr]では SFP2 の低下量が大きくなる。燃料損傷が発生した後は消防車 1 の起動率の方が大きいため、燃料損傷頻度の低下量は、400 [hr]付近で逆転する。

以下の考察に基づき、単一の使用済燃料プールの考察と同様の理由で、水位、累積停止確率は計算できていると考えている。

比較のため図 4-28 の燃料損傷頻度の変化量の計算結果に対して、1 万サンプルと 10 万サンプルで計算を行った CMMC 法のみでの計算結果を追加した図を、SFP1 については図 4-29、SFP2 については図 4-30 にそれぞれ示す。

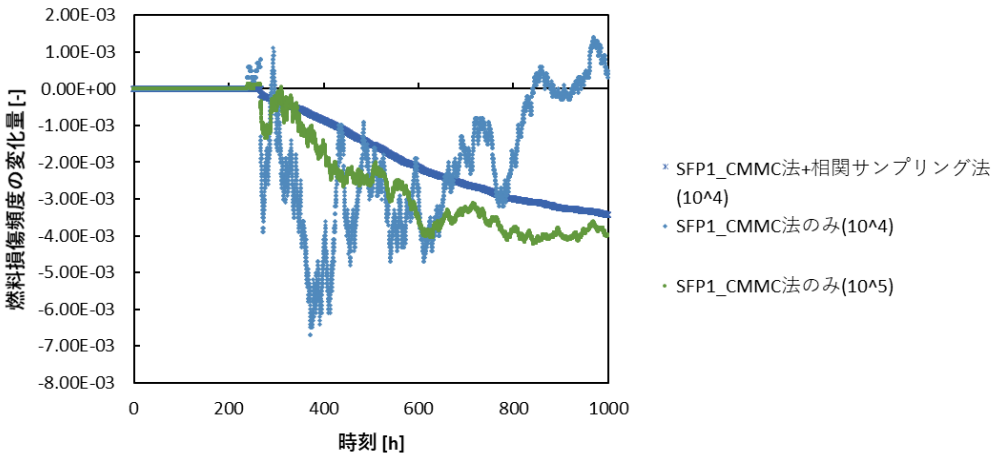


図 4-29　SFP1 の燃料損傷頻度の変化量(CMMC 法のみでの計算結果あり)

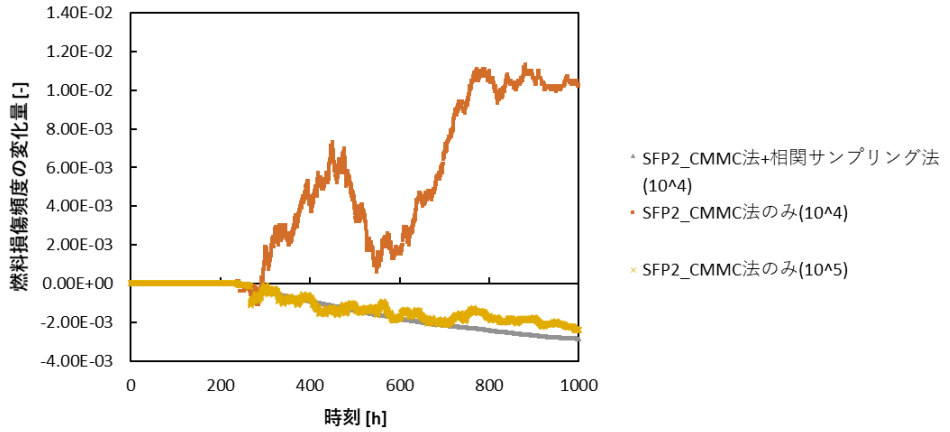


図 4-30　SFP2 の燃料損傷頻度の変化量(CMMC 法のみでの計算結果あり)

図 4-29、図 4-30 から、10 万サンプルでの CMMC 法のみの計算結果においても統計誤差の影響で振動していることがわかる。単一の SFP を対象とした解析と同様に、CMMC 法のみで計算する場合にはより多くのサンプル数を用意する必要があるといえる。また、このことから、少ないサンプル数で計算を行うことが可能である相関サンプリング法の有効性が確認できる。

4.5　本章のまとめ

本節では、本章のまとめを述べる。

- 4.1 節では、本章で CMMC 法と相関サンプリング法を用いた使用済燃料プールを対象とした感度解析の概要について説明することを述べ、本章の概要を示した。
- 4.2 節では、使用済燃料プールの解析モデルと解析条件について説明した。使用済燃料プールで想定する事故シーケンスについて説明し、使用済燃料プールを簡易化した断熱近似に基づくモデルを示した。
- 4.3 節では、単一の使用済燃料プールを対象とした解析の手順、条件、結果を示した。温度や水位、システムの状態、機器の状態、SFP 燃料損傷頻度の計算結果を示し、解析可能であることを確認した。解析解との比較が行えていないため、今後の課題として解析解の導出を考えている。解析解を求める方法については現在検討中であるが、事故シナリオに合わせて燃料損傷を考える必要があるため、現時点では導出が困難であると考えている。
- 4.4 節では、複数の使用済燃料プールを対象とした解析の手順、条件、結果を示した。温度や水位、システムの状態、機器の状態、SFP 燃料損傷頻度の計算結果を示し、SFP が二つある複雑なシステムであっても解析可能であることを確認した。単一のシステム同様、解析解を求め妥当性を確認する必要があると考えている。

参考文献

[1] 澤田憲人, “マルチユニットリスク評価への連続マルコフ過程モンテカルロ法の適用,” 修士論文, 名古屋大学, (2021).

[2] EPRI, “Modular Accident Analysis Program 5 (MAAP5) Applications Guidance: Desktop Reference for Using MAAP5 Software-Phase 3 Report,” (2017).

[3] 日本機械学会（編）, “日本機械学会蒸気表,” 日本機械学会, (1999).

[4] S. G. Popov, J. J. Carbajo, V. K. Ivanov, *et al*, “Thermophysical Properties of MOX and UO 2 Fuels Including the Effects of Irradiation,” ORNL/TM-2000/351, Oak Ridge National Laboratory, (2000).

[5] J. H. Keenan, F. G. Keyes, P. G. Hill, *et al*, “Steam Tables: Thermodynamic Properties of Water Including Vapor, Liquid, and Solid Phases International System of Units S.I.,” Wiley, (1978).

[6] ステンレス協会, “ステンレスの導電率、透磁率、熱膨張率などの物理的性質について,” http://www.jssa.gr.jp/contents/faq-article/q6/, （2021 年 12 月 23 日アクセス）.

5　結論

本章では、本論文のまとめと今後の課題を述べる。

5.1　まとめ

本節では、本論文のまとめを述べる。

第 1 章では、本研究の背景と目的について述べた。2011 年に起きた東京電力福島第一原子力発電所事故(1F)事故を契機として、複数の原子炉ユニットが関与するリスク評価について関心が高まっていることを述べた。確率論的リスク評価(PRA)においては、イベントツリーを用いる手法が主に用いられているが、マルチユニットリスク評価のために必要となる時間依存性や相互依存性の考慮が困難である。そこで、それらの依存性を考慮に入れることが可能である連続マルコフ過程モンテカルロ(CMMC)法に着目することを述べた。CMMC 法のマルチユニットリスク評価への適用性は先行研究によって確認されているが、CMMC 法を適用したときに発生する統計誤差の影響で故障率の変動などが解析結果に与える影響を評価する感度解析が困難であること述べた。このような背景から、本研究では感度解析を行う際の統計誤差を減らすための検討として、相関サンプリング法を用いてリスクアセスメントにおける感度解析を行うことを目的とした。本研究の目的は、小目的を順に達成することで達成することが出来る。小目的は「CMMC 法への相関サンプリング法の適用の検討」、「相関サンプリング法を、用いた単純なシステムに対する感度解析の妥当性確認」、「相関サンプリング法を用いた現実的なモデルを対象とした感度解析」の三つである。

第 2 章では、確率論的リスク評価の感度解析について述べた。PRA 手法として、イベントツリーを用いる手法と CMMC 法を説明した。また、リスク重要度指数を用いるリスク評価について説明した。その後、PRA における感度解析を行う際、イベントツリーを用いる手法では計算結果に統計誤差が生じないが、CMMC 法ではモンテカルロ法を用いることから計算結果に大きな統計誤差が含まれるため感度を評価することが難しいことを説明した。さらに、CMMC 法に相関サンプリング法を適用するため、ウエイトの導出と適用方法について理解を深めた。

第 3 章では、単純なシステムを対象とした感度解析について述べた。CMMC 法と第 2 章で説明した相関サンプリング法を用いて単純なシステムを対象とした感度解析を行い、解析解と比較した。単純なシステムとして、一つの機器を持つシステムと二つの機器を持つシステムの二つのシステムを解析した。二つの機器を持つシステムは機器の故障に相互依存性がある場合を想定した。解析結果から、CMMC 法と相関サンプリング法組み合わせた計算結果は解析解と一致していることを確認した。また、CMMC 法のみでの計算結果は統計誤差のため解析解と差異が生じることも確認した。単純なシステムを対象とした感度解析を行うことが可能であることを確認したことで、より現実的なシステムである使用済燃料プールの解析に進んだ。

第 4 章では、使用済燃料プールを対象とした感度解析について述べた。使用済燃料プー

ルの解析を行うため、断熱近似に基づく解析モデルを作成した。その後、作成した解析モデルと CMMC 法、相関サンプリング法を用いて、単一の使用済燃料プールを対象とした解析と複数の使用済燃料プール間の依存性を考慮した解析を行った。解析結果として、温度や水位、システムの状態、機器の状態、燃料損傷頻度の計算結果を示し、考察を行った。

第 2 章にて、CMMC 法への相関サンプリング法の適用の検討をし、第 3 章にて、相関サンプリング法を用いた単純なシステムに対する感度解析の妥当性を確認し、第 4 章にて、相関サンプリング法を用いた現実的なモデルを対象とした感度解析を行った。以上の小目的を達成したため、本研究の目的である相関サンプリング法を用いて、CMMC 法における感度解析手法の検討を達成することが出来たと考えている。

5.2 今後の課題

本節では、今後の課題を述べる。課題として以下の二つを挙げる。

- 人的因子による感度解析

本論文においては、機器の故障率や起動率を摂動させる感度解析を行ったが、過酷事故時の対応には人的因子も関わってくる。そのため、人的因子に対しても感度解析を行っていく必要がある。人的因子は、操作する場所、操作のしやすさ、人のストレス度や熟練度、他の人によるリカバリのしやすさなど、多くの要因による影響を受けるため、機械の故障率や起動率と違って、評価することが困難である。人的因子の定量的な評価方法を検討中である。

- 実機の Severe accident management への適用

第 4 章にて、使用済燃料プールの感度解析を行ったが、用いたパラメータは相関サンプリング法の適用性を確認するための仮想的なパラメータであった。第 4 章の検討にて相関サンプリング法の適用性を確認したため、仮想的なパラメータを実機で想定されるパラメータに変更して感度解析を行うことを検討している。

Appendix. A　解析に用いた計算コードの説明

A.1 一つの機器を持つシステムの解析

CMMC 法と相関サンプリング法を用いた一つの機器を持つシステムの解析コードを示す。プログラミング言語は python を用いた。

A.1.1　1sys_CMMC_cor_hp_ver5.py

```
#! /usr/bin/env python3


from math import exp, log
import os
import random
#from matplotlib.lines import _LineStyle
import matplotlib.pyplot as plt
import numpy as np
from pandas.core.frame import DataFrame
from tqdm import tqdm
from joblib import Parallel, delayed
import function
#import emf



sample = 100000#サンプル数
failure_rate = 0.01
failure_rate_up = [0,0.1,1,10]#[%]
failure_rate_up_num = 2
Time = 100
z_0 =1.96#95%信頼区間
Cumulative_probability =[]
C_p_err =[]
t_step =0.1
t = []
x=0
while(x<Time):
    t.append(x)
    x += t_step
A_sol = []
```

```
random.seed(20210607)


#CMMC
state1_base_avg = [0]*len(t)
state1_per_avg = [0]*len(t)
state1_dif_avg = [0]*len(t)


def Make_scenario():#CMMC 法と相関サンプリング法
    components = [[0]]
    weight = [1.0]
    state_base = [[0] * (2 ** len(components))]
    state_base[0][function.Components_to_state(components[0])] = 1   #state = 0 at [0], state
= 1 at [1]
    state_per = [[0.0] * (2 ** len(components))]
    state_per[0][function.Components_to_state(components[0])] = 1.0
    for i in range(len(t)):
        if state_base[i][0] == 1:
            result = function.Determination_states(components[i], weight[i], [failure_rate],
[failure_rate_up[failure_rate_up_num]/100], t_step, state_per[i])
            components.append(result[0])
            state_base.append(result[1])
            state_per.append(result[2])
            weight.append(result[3])


        elif state_base[i][1] == 1:
            result = function.Determination_states(components[i], weight[i], [0.0], [0.0/100],
t_step, state_per[i])
            components.append(result[0])
            state_base.append(result[1])
            state_per.append(result[2])
            weight.append(result[3])


        state1_base_avg[i] += state_base[i][1] / sample
        state1_per_avg[i] += state_per[i][1] / sample
        state1_dif_avg[i] += state_per[i][1] / sample - state_base[i][1] / sample
```

```
#CMMC 法で摂動のシナリオを計算
state1_CMMC_per_avg = [0]*len(t)
state1_CMMC_dif_avg = [0]*len(t)

def CMMC_per():
    components = [[0]]
    state_per = [[0.0] * (2 ** len(components))]
    state_per[0][function.Components_to_state(components[0])] = 1
    for i in range(len(t)):
        if state_per[i][0] == 1:
            result = function.Determination_states(components[i], 1, [failure_rate *
(1+failure_rate_up[failure_rate_up_num]/100)], [failure_rate_up[failure_rate_up_num]/100],
t_step, state_per[i])
            components.append(result[0])
            state_per.append(result[1])

        elif state_per[i][1] == 1:
            result = function.Determination_states(components[i], 1, [0.0], [0.0/100], t_step,
state_per[i])
            components.append(result[0])
            state_per.append(result[1])

        state1_CMMC_per_avg[i] += state_per[i][1] / sample

for i in tqdm(range(sample),ncols=80,desc="CMMC with correlated sampling"):
    Make_scenario()

for i in tqdm(range(sample),ncols=80,desc="CMMC only"):
    CMMC_per()
for i in range(len(t)):
    state1_CMMC_dif_avg[i] = state1_CMMC_per_avg[i] - state1_base_avg[i]

time = np.arange(0,Time+0.1,0.1)
Pro0 = 1-np.exp(-time*failure_rate*(1+failure_rate_up[0]/100))
Pro01 = 1-np.exp(-time*failure_rate*(1+failure_rate_up[1]/100))
```

```
Pro1 = 1-np.exp(-time*failure_rate*(1+failure_rate_up[2]/100))
Pro10 = 1-np.exp(-time*failure_rate*(1+failure_rate_up[3]/100))
Pro_dif = 1-np.exp(-time*failure_rate*(1+failure_rate_up[failure_rate_up_num]/100))-Pro0
#plt.errorbar(t,state1_base_avg,yerr=state1_base_avg_err)
#plt.errorbar(t,state1_per_avg,yerr=state1_per_avg_err)
plt.plot(t,state1_base_avg,label="CMMC_base",marker="o",markersize=1)
plt.plot(t,state1_per_avg,label="CMMC_per",marker="o",markersize=1)
#plt.plot(t,state1_dif_avg,label="CMMC",marker="o",markersize=1)
plt.plot(time,Pro0,label="0%_inc")
plt.plot(time,Pro01,label="0.1%_inc")
#plt.plot(time,Pro1,label="1%_inc")
#plt.plot(time,Pro10,label="10%_inc")
#plt.plot(time,Pro_dif,label="analitical")
plt.title("{}%_inc".format(failure_rate_up[failure_rate_up_num]))
plt.xlabel("t[s]")
plt.ylabel("Cumulative failure rate [-]")
plt.legend()
plt.savefig("{}%_inc.png".format(failure_rate_up[failure_rate_up_num]))
plt.show()



def Ruiseki_inf1():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def Ruiseki_inf3():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

def Ruiseki_base_plt():
    plt.plot(t,Pro0,color="green",label="Analytical",linestyle="--")
```

```
    plt.plot(time,state1_base_avg,color="green",label="CMMC")
    Ruiseki_inf1()

def Ruiseki_per_plt():
    plt.plot(t,Pro1,color="green",label="Analytical",linestyle="--")
    plt.plot(time,state1_per_avg,color="green",label="CMMC with correlated sampling")
    Ruiseki_inf1()

#def Ruiseki_per_plt():
#    plt.plot(t,eA_per,color="green",label="Component A_Analytical_per",linestyle="--")
#    plt.plot(t,eB_per,color="orange",label="Component B_Analytical_per",linestyle="--")
#    plt.plot(t,A_state1_per_avg,color="green",label="Component A_CMMC_per")Difference of
cumulative failure probability
#    plt.plot(t,B_state1_per_avg,color="orange",label="Component B_CMMC_per")
#    Ruiseki_inf1()

def Ruiseki_dif_plt():
    plt.plot(t,Pro_dif,color="green",label="Analytical (perturbation - base)",lw = 3)
    plt.plot(t,state1_CMMC_dif_avg,color="blue",label="CMMC only (perturbation - base)")
    #plt.plot(t,state1_dif_avg,color="red",label="CMMC with correlated sampling (perturbation
- base)",lw = 3, linestyle=":")
    Ruiseki_inf3()

#Ruiseki_base_plt()
#Ruiseki_per_plt()
#Ruiseki_dif_plt()

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2',
'#7f7f7f', '#bcbd22', '#17becf']

def states_per_plt_ver1():
    y = [Pro1, state1_per_avg,]
    for i in range(len(y)):
        if i%2 == 0:
            plt.plot(t, y[i], color=colors[i//2], lw=5, zorder=1)
        elif i%2 == 1:
```

```
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            plt.scatter(t, y[i], marker="o", edgecolors=colors[i//2], linewidth=0.2,
c="white", s=12, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def states_dif_plt_ver1():
    y = [Pro_dif, state1_CMMC_dif_avg, state1_dif_avg,]
    for i in range(len(y)):
        if i%3 == 0:
            plt.plot(t, y[i], color=colors[i//3], lw=3, zorder=1)
        elif i%3 == 1:
            plt.plot(t, y[i], linestyle="--", markeredgecolor=colors[i//2], markersize=2)
            #plt.scatter(t, y[i], marker="D", c=colors[i//3], s=3, zorder=3)
        elif i%3 == 2:
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            #plt.scatter(t, y[i], marker="o", edgecolors=colors[i//3], linewidth=0.2,
c="white", s=12, zorder=2)
            plt.scatter(t[::30], y[i][::30], marker="o", edgecolors="black", linewidth=2,
c=colors[i//3], s=25, zorder=2)
            #plt.scatter(t, y[i], marker="d", edgecolors="black", linewidth=0.4,
c=colors[i//3], s=40, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

#states_per_plt_ver1()
states_dif_plt_ver1()
```

A.1.2 function.py

```
import numpy as np
import random
```

```
def Make_weight(weight_before, fairule_rate, Perturbation_of_failure_rate, Time_step_width,
changed_components):
    weight_after = weight_before
    for i in range(len(fairule_rate)):
        weight_after = weight_after * np.exp(- fairule_rate[i] *
Perturbation_of_failure_rate[i] * Time_step_width)
        if changed_components[i] == 1:
            weight_after = weight_after * (1 + Perturbation_of_failure_rate[i])
    return weight_after


#print(Make_weight(1.01, [0.2, 0.3, 0.5], [0.01, 0.02, -0.01], 0.1, [1, 0, 1]))
#print(Make_weight(1.0, [0.01], [0.01], 0.1, [0]))


def Components_to_state(components):
    state = 0
    for i in range(len(components)):
        state += components[i] * 2**i
    return state


#print(Components_to_state([1, 0, 1]))


def Determination_states(
    components,
    weight_before,
    fairule_rate,
    Perturbation_of_failure_rate,
    Time_step_width,
    state_per_before):
    ran = []
    state_base_before = [0] * (2 ** len(components))
    state_base_before[Components_to_state(components)] = 1
    state_base_after = [0] * (2 ** len(components))
    state_per_after = [0] * (2 ** len(components))
    changed_components = [0] * len(components)
    for i in range (len(components)):
        ran.append(random.random())
```

```
        if ran[i] < fairule_rate[i] * Time_step_width:
            changed_components[i] = 1
            if components[i] == 0:
                components[i] = 1
            else:
                components[i] = 0
    weight_after = Make_weight(
        weight_before,
        fairule_rate,
        Perturbation_of_failure_rate,
        Time_step_width,
        changed_components)
    state_base_after[Components_to_state(components)] = 1
    for i in range(len(state_base_before)):
        state_per_after[i] = state_per_before[i] + (state_base_after[i] -
state_base_before[i]) * weight_after
    return components, state_base_after, state_per_after, weight_after


"""
システムに要素が 3 つ
その時間での故障率や摂動割合を入力
components:要素の状態  ex.[0, 0, 1]
weight_before:時間変化有りの前ステップでのウエイト  ex.1.01
Fairule_rate:故障率 λ    ex.[0.2, 0.3, 0.5]
Perturbation_of_failure_rate:摂動割合   ex.[0.01, 0.02, -0.01]
Time_step_width:時間幅 Δt    ex.0.1
changed_components:要素の状態が変わるものを 1 とする(状態が変わらないときすべて 0)  ex.[1, 0, 1]
weight_after:時間変化有りの weight]
"""
```

A.2 二つの機器を持つシステムの解析

CMMC 法と相関サンプリング法を用いた二つの機器を持つシステムの解析コードを示す。こちらも同様、プログラミング言語は python を用いた。

A.2.1 easysys_CMMC_cor_hp_ver9.py

```
#! /usr/bin/env python3


from datetime import time

from math import exp, log

import random

import matplotlib.pyplot as plt

import numpy as np

from pandas.core.frame import DataFrame

from tqdm import tqdm

from numba import jit, i8

from joblib import Parallel, delayed

import function


c1A_base = {"Bo":0.2,"Bx":0.8}

c1A_per_rate = {"Bo":1.0,"Bx":-2.0}# [%]

c1B_base = {"Ao":0.1,"Ax":0.01}

c1B_per_rate = {"Ao":3.0,"Ax":4.0}# [%]


c2 = 0.001

Time = 100

t_step = 0.1

num = [100,1000,10000,100000,1000000]

num_num = 1


def PX(c1X,t):

    return c1X*(1-exp(-c2*t))


c1A_per =

{"Bo":c1A_base["Bo"]*(1+c1A_per_rate["Bo"]/100),"Bx":c1A_base["Bx"]*(1+c1A_per_rate["Bx"]/100)

}#CMMC には用いない

c1A = {"base":c1A_base,"per":c1A_per}

c1B_per =

{"Ao":c1B_base["Ao"]*(1+c1B_per_rate["Ao"]/100),"Ax":c1B_base["Ax"]*(1+c1B_per_rate["Ax"]/100)

}#CMMC には用いない

c1B = {"base":c1B_base,"per":c1B_per}
```

```
t = []
x=0
while(x<Time):
    t.append(x)
    x += t_step

#解析的
state1_base = [1.0]
state2_base = [0.0]
state3_base = [0.0]
state4_base = [0.0]
compA_base = [0.0]
compB_base = [0.0]
state1_per = [1.0]
state2_per = [0.0]
state3_per = [0.0]
state4_per = [0.0]
compA_per = [0.0]
compB_per = [0.0]
compA_dif = [0.0]
compB_dif = [0.0]

state1_dif_anal = [0.0]
state2_dif_anal = [0.0]
state3_dif_anal = [0.0]
state4_dif_anal = [0.0]
def P1_4(t,A):
    return PX(c1A[A]["Bo"],t)*t_step*PX(c1B[A]["Ao"],t)*t_step
def P1_2(t,A):
    return PX(c1A[A]["Bo"],t)*t_step-P1_4(t,A)
def P1_3(t,A):
    return PX(c1B[A]["Ao"],t)*t_step-P1_4(t,A)
def P2_4(t,A):
```

```
    return PX(c1B[A]["Ax"],t)*t_step
def P3_4(t,A):
    return PX(c1A[A]["Bx"],t)*t_step


for i,x in tqdm(enumerate(t),ncols=80,desc="analysis"):
    if i !=1000:
        state1_base.append(state1_base[i]*(1.0-
(P1_2(x,"base")+P1_3(x,"base")+P1_4(x,"base"))))
        state2_base.append(state1_base[i]*P1_2(x,"base")+state2_base[i]*(1.0-P2_4(x,"base")))
        state3_base.append(state1_base[i]*P1_3(x,"base")+state3_base[i]*(1.0-P3_4(x,"base")))
        state4_base.append(state1_base[i]*P1_4(x,"base")+state2_base[i]*P2_4(x,"base")+state3_
base[i]*P3_4(x,"base")+state4_base[i]*1.0)
        compA_base.append(state2_base[i]+state4_base[i])
        compB_base.append(state3_base[i]+state4_base[i])

        state1_per.append(state1_per[i]*(1.0-(P1_2(x,"per")+P1_3(x,"per")+P1_4(x,"per"))))
        state2_per.append(state1_per[i]*P1_2(x,"per")+state2_per[i]*(1.0-P2_4(x,"per")))
        state3_per.append(state1_per[i]*P1_3(x,"per")+state3_per[i]*(1.0-P3_4(x,"per")))
        state4_per.append(state1_per[i]*P1_4(x,"per")+state2_per[i]*P2_4(x,"per")+state3_per[i
]*P3_4(x,"per")+state4_per[i]*1.0)
        compA_per.append(state2_per[i]+state4_per[i])
        compB_per.append(state3_per[i]+state4_per[i])

        compA_dif.append((state2_per[i]+state4_per[i])-(state2_base[i]+state4_base[i]))
        compB_dif.append((state3_per[i]+state4_per[i])-(state3_base[i]+state4_base[i]))

        state1_dif_anal.append(state1_per[i]-state1_base[i])
        state2_dif_anal.append(state2_per[i]-state2_base[i])
        state3_dif_anal.append(state3_per[i]-state3_base[i])
        state4_dif_anal.append(state4_per[i]-state4_base[i])

#CMMC 法の解析
random.seed(20210730)

compA_base_avg = [0]*len(t)
compB_base_avg = [0]*len(t)
```

```
compA_per_avg = [0]*len(t)
compB_per_avg = [0]*len(t)

compA_dif_avg = [0]*len(t)
compB_dif_avg = [0]*len(t)

state_base_avg = [0]*4
state_per_avg = [0]*4
state_dif_avg = [0]*4
state_dif_pct = [0]*4
compA_dif_pct =[0]*4
compB_dif_pct =[0]*4

for i in range(4):
    state_base_avg[i] = np.zeros(len(t))
    state_per_avg[i] = np.zeros(len(t))
    state_dif_avg[i] = np.zeros(len(t))
    state_dif_pct[i] = np.zeros(len(t))
    compA_dif_pct[i] = np.zeros(len(t))
    compB_dif_pct[i] = np.zeros(len(t))

def Make_scenario():#シナリオを作成し平均値に代入
    components = [[0, 0]]
    weight = [1.0]
    state_base = [[0] * (2 ** len(components[0]))]
    state_base[0][function.Components_to_state(components[0])] = 1   #state = 0 at [0, 0 ],
state = 1 at [1, 0], ···
    state_per = [[0.0] * (2 ** len(components[0]))]
    state_per[0][function.Components_to_state(components[0])] = 1.0
    for i in range(len(t)):
        if state_base[i][0] == 1:
            result = function.Determination_states(
                components[i],
                weight[i],
                [PX(c1A["base"]["Bo"],t[i]), PX(c1B["base"]["Ao"],t[i])],
```

```
            [c1A_per_rate["Bo"]/100, c1B_per_rate["Ao"]/100],
            t_step,
            state_per[i])

    elif state_base[i][1] == 1:
        result = function.Determination_states(
            components[i],
            weight[i],
            [0.0, PX(c1B["base"]["Ax"],t[i])],
            [0.0/100, c1B_per_rate["Ax"]/100],
            t_step,
            state_per[i])

    elif state_base[i][2] == 1:
        result = function.Determination_states(
            components[i],
            weight[i],
            [PX(c1A["base"]["Bx"],t[i]), 0.0],
            [c1A_per_rate["Bx"]/100, 0.0/100],
            t_step,
            state_per[i])

    elif state_base[i][3] == 1:
        result = function.Determination_states(
            components[i],
            weight[i],
            [0.0, 0.0],
            [0.0/100, 0.0/100],
            t_step,
            state_per[i])

    components.append(result[0])
    state_base.append(result[1])
    state_per.append(result[2])
    weight.append(result[3])
```

```
        state_base_avg[0][i] += state_base[i][0] / num[num_num]
        state_base_avg[1][i] += state_base[i][1] / num[num_num]
        state_base_avg[2][i] += state_base[i][2] / num[num_num]
        state_base_avg[3][i] += state_base[i][3] / num[num_num]

        state_per_avg[0][i] += state_per[i][0] / num[num_num]
        state_per_avg[1][i] += state_per[i][1] / num[num_num]
        state_per_avg[2][i] += state_per[i][2] / num[num_num]
        state_per_avg[3][i] += state_per[i][3] / num[num_num]

        state_dif_avg[0][i] += (state_per[i][0] - state_base[i][0]) / num[num_num]
        state_dif_avg[1][i] += (state_per[i][1] - state_base[i][1]) / num[num_num]
        state_dif_avg[2][i] += (state_per[i][2] - state_base[i][2]) / num[num_num]
        state_dif_avg[3][i] += (state_per[i][3] - state_base[i][3]) / num[num_num]

        compA_base_avg[i] += (state_base[i][1] + state_base[i][3]) / num[num_num]
        compB_base_avg[i] += (state_base[i][2] + state_base[i][3]) / num[num_num]

        compA_per_avg[i] += (state_per[i][1] + state_per[i][3]) / num[num_num]
        compB_per_avg[i] += (state_per[i][2] + state_per[i][3]) / num[num_num]

        compA_dif_avg[i] += ((state_per[i][1] + state_per[i][3]) - (state_base[i][1] +
state_base[i][3])) / num[num_num]
        compB_dif_avg[i] += ((state_per[i][2] + state_per[i][3]) - (state_base[i][2] +
state_base[i][3])) / num[num_num]

#CMMC 法で摂動のシナリオを計算
compA_CMMC_per_avg = [0]*len(t)
compB_CMMC_per_avg = [0]*len(t)

compA_CMMC_dif_avg = [0]*len(t)
compB_CMMC_dif_avg = [0]*len(t)

compA_CMMC_dif_pct = [0]*len(t)
compB_CMMC_dif_pct = [0]*len(t)
```

```
state1_CMMC_per_avg = [0]*len(t)
state2_CMMC_per_avg = [0]*len(t)
state3_CMMC_per_avg = [0]*len(t)
state4_CMMC_per_avg = [0]*len(t)

state1_CMMC_dif_avg = [0]*len(t)
state2_CMMC_dif_avg = [0]*len(t)
state3_CMMC_dif_avg = [0]*len(t)
state4_CMMC_dif_avg = [0]*len(t)

state_CMMC_dif_pct = [0]*4

for i in range(4):
    state_CMMC_dif_pct[i] = np.zeros(len(t))

def CMMC_per():
    components = [[0, 0]]
    state_per = [[0.0] * (2 **len(components[0]))]
    state_per[0][function.Components_to_state(components[0])] = 1
    for i in range(len(t)):
        if state_per[i][0] == 1:
            result = function.Determination_states(
                components[i],
                1,
                [PX(c1A["base"]["Bo"],t[i]) * (1+c1A_per_rate["Bo"]/100),
PX(c1B["base"]["Ao"],t[i]) * (1+c1B_per_rate["Ao"]/100)],
                [c1A_per_rate["Bo"]/100, c1B_per_rate["Ao"]/100],
                t_step,
                state_per[i])

        elif state_per[i][1] == 1:
            result = function.Determination_states(
                components[i],
                1,
                [0.0, PX(c1B["base"]["Ax"],t[i]) * (1+c1B_per_rate["Ax"]/100)],
                [0.0/100, c1B_per_rate["Ax"]/100],
```

```
                t_step,
                state_per[i])

        elif state_per[i][2] == 1:
            result = function.Determination_states(
                components[i],
                1,
                [PX(c1A["base"]["Bx"],t[i]) * (1+c1A_per_rate["Bx"]/100), 0.0],
                [c1A_per_rate["Bx"]/100, 0.0/100],
                t_step,
                state_per[i])

        elif state_per[i][3] == 1:
            result = function.Determination_states(
                components[i],
                1,
                [0.0, 0.0],
                [0.0/100, 0.0/100],
                t_step,
                state_per[i])

        components.append(result[0])
        state_per.append(result[1])

        state1_CMMC_per_avg[i] += state_per[i][0] / num[num_num]
        state2_CMMC_per_avg[i] += state_per[i][1] / num[num_num]
        state3_CMMC_per_avg[i] += state_per[i][2] / num[num_num]
        state4_CMMC_per_avg[i] += state_per[i][3] / num[num_num]

        compA_CMMC_per_avg[i] += (state_per[i][1] + state_per[i][3]) / num[num_num]
        compB_CMMC_per_avg[i] += (state_per[i][2] + state_per[i][3]) / num[num_num]

for i in tqdm(range(num[num_num]),ncols=80,desc="CMMC"):
    Make_scenario()
for i in range(len(t)):
    state_dif_pct[0][i] = state_dif_avg[0][i] / state1_dif_anal[i] * 100
```

```
    state_dif_pct[1][i] = state_dif_avg[1][i] / state2_dif_anal[i] * 100
    state_dif_pct[2][i] = state_dif_avg[2][i] / state3_dif_anal[i] * 100
    state_dif_pct[3][i] = state_dif_avg[3][i] / state4_dif_anal[i] * 100

    compA_dif_pct[i] = compA_dif_avg[i] / compA_dif[i] * 100
    compB_dif_pct[i] = compB_dif_avg[i] / compB_dif[i] * 100

for i in tqdm(range(num[num_num]),ncols=80,desc="CMMC only"):
    CMMC_per()
for i in range(len(t)):
    state1_CMMC_dif_avg[i] = state1_CMMC_per_avg[i] - state_base_avg[0][i]
    state2_CMMC_dif_avg[i] = state2_CMMC_per_avg[i] - state_base_avg[1][i]
    state3_CMMC_dif_avg[i] = state3_CMMC_per_avg[i] - state_base_avg[2][i]
    state4_CMMC_dif_avg[i] = state4_CMMC_per_avg[i] - state_base_avg[3][i]

    compA_CMMC_dif_avg[i] = compA_CMMC_per_avg[i] - compA_base_avg[i]
    compB_CMMC_dif_avg[i] = compB_CMMC_per_avg[i] - compB_base_avg[i]

    state_CMMC_dif_pct[0][i] = state1_CMMC_dif_avg[i] / state1_dif_anal[i] * 100
    state_CMMC_dif_pct[1][i] = state2_CMMC_dif_avg[i] / state2_dif_anal[i] * 100
    state_CMMC_dif_pct[2][i] = state3_CMMC_dif_avg[i] / state3_dif_anal[i] * 100
    state_CMMC_dif_pct[3][i] = state4_CMMC_dif_avg[i] / state4_dif_anal[i] * 100

    compA_CMMC_dif_pct[i] = compA_CMMC_dif_avg[i] / compA_dif[i] * 100
    compB_CMMC_dif_pct[i] = compB_CMMC_dif_avg[i] / compB_dif[i] * 100

def Ruiseki_inf1():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def Ruiseki_inf2():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Probability [-]")
```

```
    plt.show()


def Ruiseki_inf3():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()


def Ruiseki_inf4():
    plt.legend()
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of probability [-]")
    plt.show()


def components_base_plt():
    plt.plot(t,compA_base,color="green",label="Component A_Analytical",linestyle="--")
    plt.plot(t,compB_base,color="orange",label="Component B_Analytical",linestyle="--")
    plt.plot(t,compA_base_avg,color="green",label="Component A_CMMC")
    plt.plot(t,compB_base_avg,color="orange",label="Component B_CMMC")
    Ruiseki_inf1()


def components_per_plt():
    plt.plot(t,compA_per,color="green",label="Component A_Analytical",linestyle="--")
    plt.plot(t,compB_per,color="orange",label="Component B_Analytical",linestyle="--")
    plt.plot(t,compA_per_avg,color="green",label="Component A_Correlated Sampling")
    plt.plot(t,compB_per_avg,color="orange",label="Component B_Correlated Sampling")
    Ruiseki_inf1()


def states_base_plt():
    plt.plot(t,state1_base,color="green",label="State 1_Analytical",linestyle="--")
    plt.plot(t,state_base_avg[0],color="green",label="State 1_CMMC")
    plt.plot(t,state2_base,color="red",label="State 2_Analytical",linestyle="--")
    plt.plot(t,state_base_avg[1],color="red",label="State 2_CMMC")
    plt.plot(t,state3_base,color="blue",label="State 3_Analytical",linestyle="--")
    plt.plot(t,state_base_avg[2],color="blue",label="State 3_CMMC")
    plt.plot(t,state4_base,color="purple",label="State 4_Analytical",linestyle="--")
```

```
    plt.plot(t,state_base_avg[3],color="purple",label="State 4_CMMC")
    Ruiseki_inf2()

def states_per_plt():
    plt.plot(t,state1_per,color="green",label="State 1_Analytical",linestyle="--")
    plt.plot(t,state_per_avg[0],color="green",label="State 1_Correlated Sampling")
    plt.plot(t,state2_per,color="red",label="State 2_Analytical",linestyle="--")
    plt.plot(t,state_per_avg[1],color="red",label="State 2_Correlated Sampling")
    plt.plot(t,state3_per,color="blue",label="State 3_Analytical",linestyle="--")
    plt.plot(t,state_per_avg[2],color="blue",label="State 3_Correlated Sampling")
    plt.plot(t,state4_per,color="purple",label="State 4_Analytical",linestyle="--")
    plt.plot(t,state_per_avg[3],color="purple",label="State 4_Correlated Sampling")
    Ruiseki_inf2()


def components_dif_plt():
    plt.plot(t,compA_dif,color="green",label="Component A_Analytical")
    plt.plot(t,compB_dif,color="orange",label="Component B_Analytical")
    plt.plot(t,compA_CMMC_dif_avg,color="green",label="Component A_CMMC only")
    plt.plot(t,compB_CMMC_dif_avg,color="orange",label="Component B_CMMC only")
    plt.plot(t,compA_dif_avg,color="green",label="Component A_Correlated Sampling",lw = 3,
linestyle=":")
    plt.plot(t,compB_dif_avg,color="orange",label="Component B_Correlated Sampling",lw = 3,
linestyle=":")
    Ruiseki_inf3()

def states_dif_plt():
    plt.plot(t,state1_dif_anal,color="green",label="State 1_Analytical")
    plt.plot(t,state1_CMMC_dif_avg,color="green",label="State 1_CMMC only")
    plt.plot(t,state_dif_avg[0],color="green",label="State 1_Correlated Sampling",lw = 3,
linestyle=":")
    plt.plot(t,state2_dif_anal,color="red",label="State 2_Analytical")
    plt.plot(t,state2_CMMC_dif_avg,color="red",label="State 2_CMMC only")
    plt.plot(t,state_dif_avg[1],color="red",label="State 2_Correlated Sampling",lw = 3,
linestyle=":")
    plt.plot(t,state3_dif_anal,color="blue",label="State 3_Analytical")
```

```
    plt.plot(t,state3_CMMC_dif_avg,color="blue",label="State 3_CMMC only")
    plt.plot(t,state_dif_avg[2],color="blue",label="State 3_Correlated Sampling",lw = 3,
linestyle=":")
    plt.plot(t,state4_dif_anal,color="purple",label="State 4_Analytical")
    plt.plot(t,state4_CMMC_dif_avg,color="purple",label="State 4_CMMC only")
    plt.plot(t,state_dif_avg[3],color="purple",label="State 4_Correlated Sampling",lw = 3,
linestyle=":")
    Ruiseki_inf4()


#components_base_plt()
#components_per_plt()
#states_base_plt()
#states_per_plt()
#components_dif_plt()
#states_dif_plt()

colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '#e377c2',
'#7f7f7f', '#bcbd22', '#17becf']

def components_per_plt_ver1():
    y = [
        compA_per, compA_per_avg,
        compB_per, compB_per_avg,]
    for i in range(len(y)):
        if i%2 == 0:
            plt.plot(t, y[i], color=colors[i//2], lw=5, zorder=1)
        elif i%2 == 1:
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            plt.scatter(t, y[i], marker="o", edgecolors=colors[i//2], linewidth=0.2,
c="white", s=12, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()
```

```
def components_dif_plt_ver1():
    y = [
        compA_dif, compA_CMMC_dif_avg, compA_dif_avg,
        compB_dif, compB_CMMC_dif_avg, compB_dif_avg]
    for i in range(len(y)):
        if i%3 == 0:
            plt.plot(t, y[i], color=colors[i//3], lw=5, zorder=1)
        elif i%3 == 1:
            plt.plot(t, y[i], marker="D", markeredgecolor=colors[i//3], markersize=2)
            #plt.scatter(t, y[i], marker="D", c=colors[i//3], s=3, zorder=3)
        elif i%3 == 2:
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            plt.scatter(t, y[i], marker="o", edgecolors=colors[i//3], linewidth=0.2,
c="white", s=12, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()


def states_per_plt_ver1():
    y = [
        state1_per, state_per_avg[0],
        state2_per, state_per_avg[1],
        state3_per, state_per_avg[2],
        state4_per, state_per_avg[3],]
    for i in range(len(y)):
        if i%2 == 0:
            plt.plot(t, y[i], color=colors[i//2], lw=5, zorder=1)
        elif i%2 == 1:
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            plt.scatter(t, y[i], marker="o", edgecolors=colors[i//2], linewidth=0.2,
c="white", s=12, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()
```

```
def states_dif_plt_ver1():
    y = [
        state1_dif_anal, state1_CMMC_dif_avg, state_dif_avg[0],
        state2_dif_anal, state2_CMMC_dif_avg, state_dif_avg[1],
        state3_dif_anal, state3_CMMC_dif_avg, state_dif_avg[2],
        state4_dif_anal, state4_CMMC_dif_avg, state_dif_avg[3],]
    for i in range(len(y)):
        if i%3 == 0:
            plt.plot(t, y[i], color=colors[i//3], lw=5, zorder=1)
        elif i%3 == 1:
            plt.plot(t, y[i], marker="D", markeredgecolor=colors[i//3], markersize=2)
            #plt.scatter(t, y[i], marker="D", c=colors[i//3], s=3, zorder=3)
        elif i%3 == 2:
            #plt.plot(t, y[i], marker="o", markeredgecolor=colors[i//2], markeredgewidth=1,
fillstyle="none", markersize=3)
            plt.scatter(t, y[i], marker="o", edgecolors=colors[i//3], linewidth=0.2,
c="white", s=12, zorder=2)
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

#components_per_plt_ver1()
#components_dif_plt_ver1()
#states_per_plt_ver1()
#states_dif_plt_ver1()

def components_dif_pct_plt():
    y = [
        compA_dif_pct, compA_CMMC_dif_pct,
        compB_dif_pct, compB_CMMC_dif_pct,]
    for i in range(len(y)):
        if i%3 == 0:
            plt.plot(t, y[i], color=colors[i//2], lw=5, zorder=1)
        elif i%2 == 1:
            plt.plot(t, y[i], marker="D", markeredgecolor=colors[i//2], markersize=2)
```

```
            #plt.scatter(t, y[i], marker="D", c=colors[i//3], s=3, zorder=3)
    plt.xlabel("Time [s]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

components_dif_pct_plt()
```

A.2.2 function.py

```
import numpy as np
import random

def Make_weight(weight_before, fairule_rate, Perturbation_of_failure_rate, Time_step_width,
changed_components):
    weight_after = weight_before
    for i in range(len(fairule_rate)):
        weight_after = weight_after * np.exp(- fairule_rate[i] *
Perturbation_of_failure_rate[i] * Time_step_width)
        if changed_components[i] == 1:
            weight_after = weight_after * (1 + Perturbation_of_failure_rate[i])
    return weight_after

#print(Make_weight(1.01, [0.2, 0.3, 0.5], [0.01, 0.02, -0.01], 0.1, [1, 0, 1]))
#print(Make_weight(1.0, [0.01], [0.01], 0.1, [0]))

def Components_to_state(components):
    state = 0
    for i in range(len(components)):
        state += components[i] * 2**i
    return state

#print(Components_to_state([1, 0, 1]))

def Determination_states(
    components,
    weight_before,
    fairule_rate,
```

```
    Perturbation_of_failure_rate,
    Time_step_width,
    state_per_before):
    ran = []
    state_base_before = [0] * (2 ** len(components))
    state_base_before[Components_to_state(components)] = 1
    state_base_after = [0] * (2 ** len(components))
    state_per_after = [0] * (2 ** len(components))
    changed_components = [0] * len(components)
    for i in range (len(components)):
        ran.append(random.random())
        if ran[i] < fairule_rate[i] * Time_step_width:
            changed_components[i] = 1
            if components[i] == 0:
                components[i] = 1
            else:
                components[i] = 0
    weight_after = Make_weight(
        weight_before,
        fairule_rate,
        Perturbation_of_failure_rate,
        Time_step_width,
        changed_components)
    state_base_after[Components_to_state(components)] = 1
    for i in range(len(state_base_before)):
        state_per_after[i] = state_per_before[i] + (state_base_after[i] -
state_base_before[i]) * weight_after
    return components, state_base_after, state_per_after, weight_after


"""
システムに要素が 3 つ
その時間での故障率や摂動割合を入力
components:要素の状態  ex.[0, 0, 1]
weight_before:時間変化有りの前ステップでのウエイト  ex.1.01
Fairule_rate:故障率 λ    ex.[0.2, 0.3, 0.5]
```

```
Perturbation_of_failure_rate:摂動割合   ex.[0.01, 0.02, -0.01]
Time_step_width:時間幅 Δt    ex.0.1
changed_components:要素の状態が変わるものを 1 とする(状態が変わらないときすべて 0)  ex.[1, 0, 1]
weight_after:時間変化有りの weight]
"""
```

A.3 単一の使用済燃料プールを対象とした解析

CMMC 法と相関サンプリング法を用いた一つの機器を持つシステムの解析コードを示す。こちらも同様、プログラミング言語は python を用いた。

A.3.1 SFP_cor_average_ver4.py

```
from os import mkdir
import SFP_model_cor_ver3 as model
import SFP_cor_data_ver2 as data
import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
from pandas import DataFrame
import csv


result = model.SFP_model(20211001)


sample = 1000


temperature_avg = np.zeros(len(result[0]))


water_level_avg = np.zeros(len(result[0]))


cd_base_avg = np.zeros(len(result[0]))
cd_per_avg = np.zeros(len(result[0]))
cd_dif_avg = np.zeros(len(result[0]))


state0_base_avg = np.zeros(len(result[0]))
state0_per_avg = np.zeros(len(result[0]))
state0_dif_avg = np.zeros(len(result[0]))
```

```
state1_base_avg = np.zeros(len(result[0]))
state1_per_avg = np.zeros(len(result[0]))
state1_dif_avg = np.zeros(len(result[0]))

state2_base_avg = np.zeros(len(result[0]))
state2_per_avg = np.zeros(len(result[0]))
state2_dif_avg = np.zeros(len(result[0]))

state3_base_avg = np.zeros(len(result[0]))
state3_per_avg = np.zeros(len(result[0]))
state3_dif_avg = np.zeros(len(result[0]))

compA_base_avg = np.zeros(len(result[0]))
compA_per_avg = np.zeros(len(result[0]))
compA_dif_avg = np.zeros(len(result[0]))

compB_base_avg = np.zeros(len(result[0]))
compB_per_avg = np.zeros(len(result[0]))
compB_dif_avg = np.zeros(len(result[0]))

for i in tqdm(range(sample),ncols=80,desc="SFP"):
    result = model.SFP_model(i)    #return time["hour"], temp, water_level, w_chg, state,
u_test, test, change
    time = result[0]
    temp = result[1]
    water_level = result[2]
    state_base = result[3]
    state_per = result[4]
    weight = result[5]
    cd = result[6]
    cd_weight = 1.0
    for j in range(len(result[0])):
        temperature_avg[j] += temp[j] / sample

        water_level_avg[j] += water_level[j] / sample
        if j !=0:
```

```
            if cd[j-1] == 0 and cd[j] == 1:
                cd_weight = weight[j]
        cd_base_avg[j] += cd[j] / sample
        cd_per_avg[j] += cd[j] * cd_weight / sample
        cd_dif_avg[j] += cd[j] * (cd_weight - 1.0) / sample

        state0_base_avg[j] += state_base[j][0] / sample
        state0_per_avg[j] += state_per[j][0] / sample
        state0_dif_avg[j] += state_per[j][0] / sample - state_base[j][0] / sample

        state1_base_avg[j] += state_base[j][1] / sample
        state1_per_avg[j] += state_per[j][1] / sample
        state1_dif_avg[j] += state_per[j][1] / sample - state_base[j][1] / sample

        state2_base_avg[j] += state_base[j][2] / sample
        state2_per_avg[j] += state_per[j][2] / sample
        state2_dif_avg[j] += state_per[j][2] / sample - state_base[j][2] / sample

        state3_base_avg[j] += state_base[j][3] / sample
        state3_per_avg[j] += state_per[j][3] / sample
        state3_dif_avg[j] += state_per[j][3] / sample - state_base[j][3] / sample

        compA_base_avg[j] += (state_base[j][1] + state_base[j][3]) / sample
        compA_per_avg[j] += (state_per[j][1] + state_per[j][3]) / sample
        compA_dif_avg[j] += (state_per[j][1] + state_per[j][3]) / sample - (state_base[j][1] +
state_base[j][3]) / sample

        compB_base_avg[j] += (state_base[j][2] + state_base[j][3]) / sample
        compB_per_avg[j] += (state_per[j][2] + state_per[j][3]) / sample
        compB_dif_avg[j] += (state_per[j][2] + state_per[j][3]) / sample - (state_base[j][2] +
state_base[j][3]) / sample


def CSV_make():
    df0 = DataFrame(dict(
            lam = data.lam,
```

```
            Perturbation_rate = data.r,
            flow_rate = data.f_rate,
            calcuration_time = data.cal_time,
            sample = sample,
            ))
    df1 = DataFrame(dict(
            temp = temperature_avg,
            water_level = water_level_avg,
            state0 = state0_base_avg,
            state1 = state1_base_avg,
            ))
    df2 = DataFrame(dict(
            state0_base = state0_base_avg,
            state0_per = state0_per_avg,
            state0_dif = state0_dif_avg,
            state1_base = state1_base_avg,
            state1_per = state1_per_avg,
            state1_dif = state1_dif_avg,
            ))
    df3 = DataFrame(dict(
            compA_base = compA_base_avg,
            compA_per = compA_per_avg,
            compA_dif = compA_dif_avg,
            cd_base = cd_base_avg,
            cd_per = cd_per_avg,
            cd_dif = cd_dif_avg,
            ))
    fname0 = "condition.csv"
    fname1 = 'data1.csv'
    fname2 = 'data2.csv'
    fname3 = 'data3.csv'
    df0.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname0)
    df1.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname1)
    df2.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname2)
    df3.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname3)
```

```
def Temperature_plt():
    plt.plot(time,temperature_avg,color="red")
    plt.xlabel("Time [h]")
    plt.ylabel("Temparature [℃]")
    plt.show()

def Water_level_plt():
    plt.ylim(0, 14)
    plt.plot(time,water_level_avg,color="blue")
    plt.xlabel("Time [h]")
    plt.ylabel("Water level [m]")
    plt.show()

def state_base_plt():
    plt.plot(time,state0_base_avg,color="green",label="State 0_CMMC")
    plt.plot(time,state1_base_avg,color="red",label="State 1_CMMC")
    plt.plot(time,state2_base_avg,color="blue",label="State 2_CMMC")
    plt.plot(time,state3_base_avg,color="orange",label="State 3_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def state_per_plt():
    plt.plot(time,state0_per_avg,color="green",label="State 0_Correlated Sampling")
    plt.plot(time,state1_per_avg,color="red",label="State 1_Correlated Sampling")
    plt.plot(time,state2_per_avg,color="blue",label="State 2_Correlated Sampling")
    plt.plot(time,state3_per_avg,color="orange",label="State 3_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def state_dif_plt():
    plt.plot(time,state0_dif_avg,color="green",label="State 0_Correlated Sampling")
    plt.plot(time,state1_dif_avg,color="red",label="State 1_Correlated Sampling")
```

```
    plt.plot(time,state2_dif_avg,color="blue",label="State 2_Correlated Sampling")
    plt.plot(time,state3_dif_avg,color="orange",label="State 3_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of probability [-]")
    plt.show()

def components_base_plt():
    plt.plot(time,compA_base_avg,color="green",label="component A_CMMC")
    plt.plot(time,compB_base_avg,color="red",label="component B_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def components_per_plt():
    plt.plot(time,compA_per_avg,color="green",label="component A_CMMC")
    plt.plot(time,compB_per_avg,color="red",label="component B_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def components_dif_plt():
    plt.plot(time,compA_dif_avg,color="green",label="component A_CMMC")
    plt.plot(time,compB_dif_avg,color="red",label="component B_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

def cd_base_plt():
    plt.plot(time,cd_base_avg,color="green",label="cd_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def cd_per_plt():
    plt.plot(time,cd_per_avg,color="green",label="cd_Correlated Sampling")
    plt.xlabel("Time [h]")
```

```
    plt.ylabel("Probability [-]")
    plt.show()

def cd_dif_plt():
    plt.plot(time,cd_dif_avg,color="green",label="cd_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of probability [-]")
    plt.show()

CSV_make()
Temperature_plt()
Water_level_plt()
state_base_plt()
state_per_plt()
state_dif_plt()
components_base_plt()
components_per_plt()
components_dif_plt()
cd_base_plt()
cd_per_plt()
cd_dif_plt()
```

A.3.2 SFP_cor_data_ver2.py

```
#! /usr/bin/env python3


cal_time = 1000  #[h]計算時間
length = {}
number = {}
weight = {}
heat = {}
s_heat = {}
f_rate = {}
lam = {}
r = {}


#SFP 形状に関するパラメータ
```

```
length["height"] = 22.58    #高さ[m]
area = 131.2    #床面積[m^2]
number["all"] = 1020    #プール内燃料体数[体](全部)
number["fuel_1"] = 255  #プール内燃料体数[体](燃料 1)
number["fuel_2"] = 765  #プール内燃料体数[体](燃料 2)
weight["Zr_cover"] = 37332  #被覆管 Zr 重量[kg]
weight["Zr_fuel"] = 33354   #燃料菅 Zr 重量[kg]
weight["Zr_support_high"] = 760  #燃料支持具上部 Zr 重量[kg]
weight["Zr_support_low"] = 760  #燃料支持具下部 Zr 重量[kg]
weight["SUS_support_high"] = 400 #燃料支持具上部ステンレス鋼重量[kg]
#weight["Zr_support"] = 760  #燃料支持具上部 Zr 重量[kg]
length["TAF"] = 4.13    #有効燃料長頂部(TAF)[m]
length["WL_control"] = 12.9 #SFP 水位制御基準水位[m]

weight["Zr"] = weight["Zr_cover"] + weight["Zr_fuel"] + weight["Zr_support_high"] +
weight["Zr_support_low"]
weight["UO2"] = 173451  #UO2 重量[kg]    170.05[kg/本]*1020[本]

#解析の初期状態に関する入力パラメータ
temp_0 = 35   #SFP 水温[℃]
length["WL_0"] = 11.9   #SFP 水位[m]
heat["fuel_1_decay"] = 7400 #燃料 1 の崩壊熱[W]
heat["fuel_2_decay"] = 1700 #燃料 2 の崩壊熱[W]

#各種の物性値
s_heat["water"] = 4.2 * 10**3   #水比熱[J/kg/K]
l_heat = 2257*10**3   #水潜熱[J/kg]
s_heat["UO2"] = 0.373 * 10**3   #UO2 比熱[J/kg/K]
s_heat["Zr"] = 0.278 * 10**3    #Zr 比熱[J/kg/K]
s_heat["SUS"] = 0.46 * 10**3    #ステンレス鋼比熱[J/kg/K]

#消化系ポンプの解析条件
f_rate["pomp1"] = 170 * 10**3    #消化系流量 [kg/h]
#l_rate = 0.005  #消化系機能喪失率 [/h]
#r_rate = 0.005  #消化系機能回復率 [/h]
lam["pomp1"] = {"l_rate":0.005, "r_rate":0.0}  #遷移率{機能喪失率, 機能回復率}
```

```
r["pomp1"] = {"l_rate":0.01, "r_rate":0.0}    #摂動割合{機能喪失率, 機能回復率}


#消防車の解析条件
f_rate["FE1"] = 160 * 10**3    #消化系流量 [kg/h]
lam["FE1"] = {"l_rate":0.0, "r_rate":0.005}   #遷移率{機能喪失率, 機能回復率}
r["FE1"] = {"l_rate":0.0, "r_rate":0.01}    #摂動割合{機能喪失率, 機能回復率}
```

A.3.3 SFP_model_cor_ver3.py

```
#! /usr/bin/env python3


from math import exp
from typing import AsyncIterable
import SFP_cor_data_ver2 as data
import matplotlib.pyplot as plt
import numpy as np
import random
import function_ver2 as function



def SFP_model(seed):
    random.seed(seed)
    time = {}
    time["second"] = [0,0.001,304]   #[s]
    time["hour"] = [0,0.001/60/60,304/60/60]
    step = (data.cal_time*60*60)//300
    for i in range(step):
        if i == 0 or i == 1 or i == 2:
            pass
        else:
            time["second"].append(time["second"][i-1]+300.0)
            time["hour"].append(time["second"][i]/60/60)


    temp = [data.temp_0]
    water_level = [data.length["WL_0"]]
    components = [[0, 1]]   #要素について
    weight = [1.0]
```

```
    state_base = [[0] * (2 ** len(components[0]))]
    state_base[0][function.Components_to_state(components[0])] = 1   #state = 0 (state_base =
[1, 0, 0]) at component = [0, 0], 1 at [1, 0], 2 at [0, 1]
    state_per = [[0.0] * (2 ** len(components[0]))]
    state_per[0][function.Components_to_state(components[0])] = 1.0
    u = function.U0()
    append_cd = 0
    cd = [0]
    dt = {}
    for j in range(len(time["second"])-1):
        dt["second"] = time["second"][j+1] - time["second"][j]
        dt["hour"] = time["hour"][j+1] - time["hour"][j]

        """ありえない
        if state_base[j][0] == 1:
            result = function.Determination_states(
                components[j], weight[j], [data.lam["pomp1"]["l_rate"]],
[data.r["pomp1"]["l_rate"]], dt["hour"], state_per[j])
        """

        if state_base[j][1] == 1:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"]["l_rate"]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"]["l_rate"]],
                dt["hour"], state_per[j])

        elif state_base[j][2] == 1:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["l_rate"], 0.0],
                [data.r["pomp1"]["l_rate"], 0.0],
                dt["hour"], state_per[j])

        elif state_base[j][3] == 1:
            result = function.Determination_states(
```

```
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"]["r_rate"]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"]["r_rate"]],
                dt["hour"], state_per[j])

        components.append(result[0])
        state_base.append(result[1])
        state_per.append(result[2])
        weight.append(result[3])

        flow_rate = function.Flow_rate(components[j], water_level[j], dt["hour"])[0]
        water_level.append(function.Flow_rate(components[j], water_level[j], dt["hour"])[1])
        water_weight = data.area * water_level[j+1] * 10**3
        u += function.U(flow_rate) - function.Qi(dt["second"])
        if u > 0:
            temp.append(
                function.Temperature_avg(temp[j], water_level[j], flow_rate)
                + function.Temperature_chg(dt["second"], water_weight))
        else:
            temp.append(100.0)
            water_level[j+1] = max(0.0, water_level[j+1] - function.Evaporation(u) * 10**(-3)
/ data.area)
            u = 0.0
        if water_level[j+1] < data.length["TAF"]:
            append_cd = 1
        cd.append(append_cd)

    return time["hour"], temp, water_level, state_base, state_per, weight, cd


#一つのサンプルの結果を表示
result = SFP_model(1200001)

def Temperature_plt():
    plt.plot(result[0], result[1], color="red")
    plt.xlabel("Time [h]")
```

```
    plt.ylabel("Temparature [℃]")
    plt.show()


def Water_level_plt():
    plt.ylim(-3, 14)
    plt.plot(result[0], result[2], color="blue")
    plt.xlabel("Time [h]")
    plt.ylabel("Water level [m]")
    plt.show()


def Temp_and_WL_plt():
    fig, ax1 = plt.subplots()
    ax1.plot(result[0], result[1], color="red")
    ax1.set_xlabel("Time [h]")
    ax1.set_ylabel("Temparature [℃]")
    ax2 = ax1.twinx()
    ax2.plot(result[0], result[2], color="blue")
    ax2.set_ylabel("water_level [m]")
    ax2.set_ylim(-3, 14)
    ax3 = ax1.twinx()
    ax3.plot(result[0], result[3], color="green")
    ax3.set_ylabel("state [-]")
    #plt.tight_layout()
    #for x,i in enumerate(state):
    #    if x != 0:
    #        ax2.axvspan(th[i],th[i+1], color= "coral")
    plt.show()


def U_plt():
    plt.plot(result[0], result[4], color="green")
    plt.xlabel("Time [h]")
    plt.ylabel("U []")
    plt.show()


#Temperature_plt()
#Water_level_plt()
```

```
#Temp_and_WL_plt()

#U_plt()
```

A.4 複数の使用済燃料プール間の依存性を考慮した解析

CMMC 法と相関サンプリング法を用いた一つの機器を持つシステムの解析コードを示す。こちらも同様、プログラミング言語は python を用いた。

A.4.1 SFP_cor_average_ver5.py

```
from os import mkdir

import SFP_model_cor_ver4 as model

import SFP_cor_data_ver3 as data

import function_ver3 as function

import matplotlib.pyplot as plt

from tqdm import tqdm

import numpy as np

from pandas import DataFrame

import csv


result = model.SFP_model(20211001)


sample = 10000


temperature_SFP1_avg = np.zeros(len(result[0]))

temperature_SFP2_avg = np.zeros(len(result[0]))


water_level_SFP1_avg = np.zeros(len(result[0]))

water_level_SFP2_avg = np.zeros(len(result[0]))


cd_SFP1_base_avg = np.zeros(len(result[0]))

cd_SFP1_per_avg = np.zeros(len(result[0]))

cd_SFP1_dif_avg = np.zeros(len(result[0]))


cd_SFP2_base_avg = np.zeros(len(result[0]))

cd_SFP2_per_avg = np.zeros(len(result[0]))

cd_SFP2_dif_avg = np.zeros(len(result[0]))
```

```
state_base_avg = [0]*(2**4)
state_per_avg = [0]*(2**4)
state_dif_avg = [0]*(2**4)

for i in range(2**4):
    state_base_avg[i] = np.zeros(len(result[0]))
    state_per_avg[i] = np.zeros(len(result[0]))
    state_dif_avg[i] = np.zeros(len(result[0]))

compA_base_avg = np.zeros(len(result[0]))
compA_per_avg = np.zeros(len(result[0]))
compA_dif_avg = np.zeros(len(result[0]))

compB_base_avg = np.zeros(len(result[0]))
compB_per_avg = np.zeros(len(result[0]))
compB_dif_avg = np.zeros(len(result[0]))

compC_base_avg = np.zeros(len(result[0]))
compC_per_avg = np.zeros(len(result[0]))
compC_dif_avg = np.zeros(len(result[0]))

compD_base_avg = np.zeros(len(result[0]))
compD_per_avg = np.zeros(len(result[0]))
compD_dif_avg = np.zeros(len(result[0]))

for i in tqdm(range(sample),ncols=80,desc="SFP"):
    result = model.SFP_model(i)    #return time["hour"], temp, water_level, w_chg, state,
u_test, test, change
    time = result[0]
    temp_SFP1 = result[1]
    temp_SFP2 = result[2]
    water_level_SFP1 = result[3]
    water_level_SFP2 = result[4]
    state_base = result[5]
    state_per = result[6]
```

```
    weight = result[7]
    cd_SFP1 = result[8]
    cd_SFP2 = result[9]
    cd_weight_SFP1 = 1.0
    cd_weight_SFP2 = 1.0
    for j in range(len(result[0])):
        temperature_SFP1_avg[j] += temp_SFP1[j] / sample
        temperature_SFP2_avg[j] += temp_SFP2[j] / sample

        water_level_SFP1_avg[j] += water_level_SFP1[j] / sample
        water_level_SFP2_avg[j] += water_level_SFP2[j] / sample
        if j !=0:
            if cd_SFP1[j-1] == 0 and cd_SFP1[j] == 1:
                cd_weight_SFP1 = weight[j]

            if cd_SFP2[j-1] == 0 and cd_SFP2[j] == 1:
                cd_weight_SFP2 = weight[j]

        cd_SFP1_base_avg[j] += cd_SFP1[j] / sample
        cd_SFP1_per_avg[j] += cd_SFP1[j] * cd_weight_SFP1 / sample
        cd_SFP1_dif_avg[j] += cd_SFP1[j] * (cd_weight_SFP1 - 1.0) / sample

        cd_SFP2_base_avg[j] += cd_SFP2[j] / sample
        cd_SFP2_per_avg[j] += cd_SFP2[j] * cd_weight_SFP2 / sample
        cd_SFP2_dif_avg[j] += cd_SFP2[j] * (cd_weight_SFP2 - 1.0) / sample
        test = 0
        for k in range(2**4):
            state_base_avg[k][j] += state_base[j][k] / sample
            state_per_avg[k][j] += state_per[j][k] / sample
            state_dif_avg[k][j] += state_per[j][k] / sample - state_base[j][k] / sample
            components = function.state_to_Components(k)
            test += state_base[j][k]
            for l, comp in enumerate(components):
                if l == 0:
                    compA_base_avg[j] += state_base[j][k] * components[l] / sample
                    compA_per_avg[j] += state_per[j][k] * components[l] / sample
```

```
                    compA_dif_avg[j] += (state_per[j][k] - state_base[j][k]) * components[l] /
sample
                if l == 1:
                    compB_base_avg[j] += state_base[j][k] * components[l] / sample
                    compB_per_avg[j] += state_per[j][k] * components[l] / sample
                    compB_dif_avg[j] += (state_per[j][k] - state_base[j][k]) * components[l] /
sample
                if l == 2:
                    compC_base_avg[j] += state_base[j][k] * components[l] / sample
                    compC_per_avg[j] += state_per[j][k] * components[l] / sample
                    compC_dif_avg[j] += (state_per[j][k] - state_base[j][k]) * components[l] /
sample
                if l == 3:
                    compD_base_avg[j] += state_base[j][k] * components[l] / sample
                    compD_per_avg[j] += state_per[j][k] * components[l] / sample
                    compD_dif_avg[j] += (state_per[j][k] - state_base[j][k]) * components[l] /
sample
        #print(test)


def CSV_make():
    df0 = DataFrame(dict(
            lam = data.lam,
            Perturbation_rate = data.r,
            flow_rate = data.f_rate,
            calcuration_time = data.cal_time,
            sample = sample,
            ))
    df1 = DataFrame(dict(
            temp_SFP1 = temperature_SFP1_avg,
            temp_SFP2 = temperature_SFP2_avg,
            water_level_SFP1 = water_level_SFP1_avg,
            water_level_SFP2 = water_level_SFP2_avg,
            ))
    df2 = DataFrame(dict(
            compA_base = compA_base_avg,
            compA_per = compA_per_avg,
```

```
            compA_dif = compA_dif_avg,
            compB_base = compB_base_avg,
            compB_per = compB_per_avg,
            compB_dif = compB_dif_avg,
            ))
    df3 = DataFrame(dict(
            compC_base = compC_base_avg,
            compC_per = compC_per_avg,
            compC_dif = compC_dif_avg,
            compD_base = compD_base_avg,
            compD_per = compD_per_avg,
            compD_dif = compD_dif_avg,
            ))
    df4 = DataFrame(dict(
            cd_SFP1_base = cd_SFP1_base_avg,
            cd_SFP1_per = cd_SFP1_per_avg,
            cd_SFP1_dif = cd_SFP1_dif_avg,
            cd_SFP2_base = cd_SFP2_base_avg,
            cd_SFP2_per = cd_SFP2_per_avg,
            cd_SFP2_dif = cd_SFP2_dif_avg,
            ))
    fname0 = "condition.csv"
    fname1 = 'data1.csv'
    fname2 = 'data2.csv'
    fname3 = 'data3.csv'
    fname4 = 'data4.csv'
    df0.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname0)
    df1.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname1)
    df2.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname2)
    df3.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname3)
    df4.to_csv("D:User/Student/Documents/卒業研究/csv_test/" + fname4)


def Temperature_plt():
    plt.plot(time,temperature_SFP1_avg,color="red",label="SFP1")
    plt.plot(time,temperature_SFP2_avg,color="blue",label="SFP2")
```

```
    plt.xlabel("Time [h]")
    plt.ylabel("Temparature [℃]")
    plt.show()


def Water_level_plt():
    plt.ylim(0, 14)
    plt.plot(time,water_level_SFP1_avg,color="red",label="SFP1")
    plt.plot(time,water_level_SFP2_avg,color="blue",label="SFP2")
    plt.xlabel("Time [h]")
    plt.ylabel("Water level [m]")
    plt.show()


def state_base_plt():
    plt.plot(time,state_base_avg[0],color="green",label="State 0_CMMC")
    plt.plot(time,state_base_avg[1],color="red",label="State 1_CMMC")
    plt.plot(time,state_base_avg[2],color="blue",label="State 2_CMMC")
    plt.plot(time,state_base_avg[3],color="orange",label="State 3_CMMC")
    plt.plot(time,state_base_avg[4],color="orange",label="State 4_CMMC")
    plt.plot(time,state_base_avg[5],color="orange",label="State 5_CMMC")
    plt.plot(time,state_base_avg[6],color="orange",label="State 6_CMMC")
    plt.plot(time,state_base_avg[7],color="orange",label="State 7_CMMC")
    plt.plot(time,state_base_avg[8],color="orange",label="State 8_CMMC")
    plt.plot(time,state_base_avg[9],color="orange",label="State 9_CMMC")
    plt.plot(time,state_base_avg[10],color="orange",label="State 10_CMMC")
    plt.plot(time,state_base_avg[11],color="orange",label="State 11_CMMC")
    plt.plot(time,state_base_avg[12],color="orange",label="State 12_CMMC")
    plt.plot(time,state_base_avg[13],color="orange",label="State 13_CMMC")
    plt.plot(time,state_base_avg[14],color="orange",label="State 14_CMMC")
    plt.plot(time,state_base_avg[15],color="orange",label="State 15_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()


def state_per_plt():
    plt.plot(time,state_per_avg[0],color="green",label="State 0_Correlated Sampling")
    plt.plot(time,state_per_avg[1],color="red",label="State 1_Correlated Sampling")
```

```
    plt.plot(time,state_per_avg[2],color="blue",label="State 2_Correlated Sampling")
    plt.plot(time,state_per_avg[3],color="orange",label="State 3_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def state_dif_plt():
    plt.plot(time,state_dif_avg[0],color="green",label="State 0_Correlated Sampling")
    plt.plot(time,state_dif_avg[1],color="red",label="State 1_Correlated Sampling")
    plt.plot(time,state_dif_avg[2],color="blue",label="State 2_Correlated Sampling")
    plt.plot(time,state_dif_avg[3],color="orange",label="State 3_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of probability [-]")
    plt.show()

def components_base_plt():
    plt.plot(time,compA_base_avg,color="green",label="component A_CMMC")
    plt.plot(time,compB_base_avg,color="red",label="component B_CMMC")
    plt.plot(time,compC_base_avg,color="blue",label="component C_CMMC")
    plt.plot(time,compD_base_avg,color="orange",label="component D_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def components_per_plt():
    plt.plot(time,compA_per_avg,color="green",label="component A_Correlated Sampling")
    plt.plot(time,compB_per_avg,color="red",label="component B_Correlated Sampling")
    plt.plot(time,compC_per_avg,color="blue",label="component C_Correlated Sampling")
    plt.plot(time,compD_per_avg,color="orange",label="component D_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Cumulative failure probability [-]")
    plt.show()

def components_dif_plt():
    plt.plot(time,compA_dif_avg,color="green",label="component A_Correlated Sampling")
    plt.plot(time,compB_dif_avg,color="red",label="component B_Correlated Sampling")
```

```
    plt.plot(time,compC_dif_avg,color="blue",label="component C_Correlated Sampling")
    plt.plot(time,compD_dif_avg,color="orange",label="component D_Correlated Sampling")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of cumulative failure probability [-]")
    plt.show()

def cd_base_plt():
    plt.plot(time,cd_SFP1_base_avg,color="green",label="SFP1")
    plt.plot(time,cd_SFP2_base_avg,color="blue",label="SFP2")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def cd_per_plt():
    plt.plot(time,cd_SFP1_per_avg,color="green",label="SFP1")
    plt.plot(time,cd_SFP2_per_avg,color="blue",label="SFP2")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()

def cd_dif_plt():
    plt.plot(time,cd_SFP1_dif_avg,color="green",label="SFP1")
    plt.plot(time,cd_SFP2_dif_avg,color="blue",label="SFP2")
    plt.xlabel("Time [h]")
    plt.ylabel("Difference of probability [-]")
    plt.show()

CSV_make()
Temperature_plt()
Water_level_plt()
state_base_plt()
state_per_plt()
state_dif_plt()
components_base_plt()
components_per_plt()
components_dif_plt()
```

```
cd_base_plt()

cd_per_plt()

cd_dif_plt()
```

A.4.2 SFP_cor_data_ver3.py

```
#! /usr/bin/env python3


cal_time = 1000  #[h]計算時間

length = {}

number = {}

weight = {}

heat = {}

s_heat = {}

f_rate = {}

lam = {}

r = {}


#SFP 形状に関するパラメータ

length["height"] = 22.58    #高さ[m]

area = 131.2    #床面積[m^2]

number["all"] = 1020    #プール内燃料体数[体](全部)

number["fuel_1"] = 255  #プール内燃料体数[体](燃料 1)

number["fuel_2"] = 765  #プール内燃料体数[体](燃料 2)

weight["Zr_cover"] = 37332  #被覆管 Zr 重量[kg]

weight["Zr_fuel"] = 33354   #燃料菅 Zr 重量[kg]

weight["Zr_support_high"] = 760  #燃料支持具上部 Zr 重量[kg]

weight["Zr_support_low"] = 760  #燃料支持具下部 Zr 重量[kg]

weight["SUS_support_high"] = 400 #燃料支持具上部ステンレス鋼重量[kg]

#weight["Zr_support"] = 760  #燃料支持具上部 Zr 重量[kg]

length["TAF"] = 4.13    #有効燃料長頂部(TAF)[m]

length["WL_control"] = 12.9 #SFP 水位制御基準水位[m]


weight["Zr"] = weight["Zr_cover"] + weight["Zr_fuel"] + weight["Zr_support_high"] +

weight["Zr_support_low"]

weight["UO2"] = 173451  #UO2 重量[kg]    170.05[kg/本]*1020[本]
```

```
#解析の初期状態に関する入力パラメータ
temp_0 = 35   #SFP 水温[℃]
length["WL_0"] = 11.9   #SFP 水位[m]
heat["fuel_1_decay"] = 7400 #燃料 1 の崩壊熱[W]
heat["fuel_2_decay"] = 1700 #燃料 2 の崩壊熱[W]

#各種の物性値
s_heat["water"] = 4.2 * 10**3   #水比熱[J/kg/K]
l_heat = 2257*10**3   #水潜熱[J/kg]
s_heat["UO2"] = 0.373 * 10**3   #UO2 比熱[J/kg/K]
s_heat["Zr"] = 0.278 * 10**3    #Zr 比熱[J/kg/K]
s_heat["SUS"] = 0.46 * 10**3    #ステンレス鋼比熱[J/kg/K]

#SFP1

#消化系ポンプ 1 の解析条件
f_rate["pomp1"] = 170 * 10**3    #消化系流量 [kg/h]
lam["pomp1"] = {"l_rate":0.005, "r_rate":0.0}  #遷移率{機能喪失率, 機能回復率}
r["pomp1"] = {"l_rate":0.01, "r_rate":0.0}    #摂動割合{機能喪失率, 機能回復率}

#消防車 1 の解析条件
f_rate["FE1"] = 160 * 10**3    #消化系流量 [kg/h]
lam["FE1"] = {"l_rate":0.0, "r_rate_SFP2_O":0.005, "r_rate_SFP2_X":0.0005}   #遷移率{機能喪失
率, 機能回復率(SFP2 燃料破損前), 機能回復率(SFP2 燃料破損後)}
r["FE1"] = {"l_rate":0.0, "r_rate_SFP2_O":0.01, "r_rate_SFP2_X":0.01}    #摂動割合{機能喪失率,
機能回復率}

#SFP2

#消化系ポンプ 2 の解析条件
f_rate["pomp2"] = 170 * 10**3    #消化系流量 [kg/h]
lam["pomp2"] = {"l_rate":0.005, "r_rate":0.0}  #遷移率{機能喪失率, 機能回復率}
r["pomp2"] = {"l_rate":0.01, "r_rate":0.0}    #摂動割合{機能喪失率, 機能回復率}

#消防車 2 の解析条件
f_rate["FE2"] = 160 * 10**3    #消化系流量 [kg/h]
```

```
lam["FE2"] = {"l_rate":0.0, "r_rate_SFP1_O":0.010, "r_rate_SFP1_X":0.0001}   #遷移率{機能喪失率, 機能回復率(SFP1 燃料破損前), 機能回復率(SFP1 燃料破損後)}
r["FE2"] = {"l_rate":0.0, "r_rate_SFP1_O":0.01, "r_rate_SFP1_X":0.01}    #摂動割合{機能喪失率, 機能回復率}
```

A.4.3 SFP_model_cor_ver4.py

```
#! /usr/bin/env python3


from math import exp
from typing import AsyncIterable
import SFP_cor_data_ver3 as data
import matplotlib.pyplot as plt
import numpy as np
import random
import function_ver3 as function



def SFP_model(seed):
    random.seed(seed)
    time = {}
    time["second"] = [0,0.001,304]   #[s]
    time["hour"] = [0,0.001/60/60,304/60/60]
    step = (data.cal_time*60*60)//300
    for i in range(step):
        if i == 0 or i == 1 or i == 2:
            pass
        else:
            time["second"].append(time["second"][i-1]+300.0)
            time["hour"].append(time["second"][i]/60/60)


    temp_SFP1 = [data.temp_0]
    temp_SFP2 = [data.temp_0]
    water_level_SFP1 = [data.length["WL_0"]]
    water_level_SFP2 = [data.length["WL_0"]]
    components = [[0, 1, 0, 1]]   #要素について
    weight = [1.0]
```

```
    state_base = [[0] * (2 ** len(components[0]))]
    state_base[0][function.Components_to_state(components[0])] = 1   #state = 0 (state_base =
[1, 0, 0]) at component = [0, 0], 1 at [1, 0], 2 at [0, 1]
    state_per = [[0.0] * (2 ** len(components[0]))]
    state_per[0][function.Components_to_state(components[0])] = 1.0
    u_SFP1 = function.U0()
    u_SFP2 = function.U0()
    append_cd_SFP1 = 0
    append_cd_SFP2 = 0
    cd_SFP1 = [0]
    cd_SFP2 = [0]
    dt = {}
    for j in range(len(time["second"])-1):
        dt["second"] = time["second"][j+1] - time["second"][j]
        dt["hour"] = time["hour"][j+1] - time["hour"][j]


        r_rate_FE1, r_rate_FE2 = function.SFP_cd_change_lam(cd_SFP1[j], cd_SFP2[j])
        if components[j] == [0, 1, 0, 1]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["l_rate"], 0.0, data.lam["pomp2"]["l_rate"], 0.0],
                [data.r["pomp1"]["l_rate"], 0.0, data.r["pomp2"]["l_rate"], 0.0],
                dt["hour"], state_per[j])


        elif components[j] == [1, 1, 0, 1]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"][r_rate_FE1],
data.lam["pomp2"]["l_rate"], 0.0],
                [data.r["pomp1"]["r_rate"], data.r["FE1"][r_rate_FE1],
data.r["pomp2"]["l_rate"], 0.0],
                dt["hour"], state_per[j])


        elif components[j] == [0, 1, 1, 1]:
            result = function.Determination_states(
                components[j], weight[j],
```

```
                [data.lam["pomp1"]["l_rate"], 0.0, data.lam["pomp2"]["r_rate"],
data.lam["FE2"][r_rate_FE2]],
                [data.r["pomp1"]["l_rate"], 0.0, data.r["pomp2"]["r_rate"],
data.r["FE2"][r_rate_FE2]],
                dt["hour"], state_per[j])

        elif components[j] == [1, 0, 0, 1]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"]["l_rate"],
data.lam["pomp2"]["l_rate"], 0.0],
                [data.r["pomp1"]["r_rate"], data.r["FE1"]["l_rate"],
data.r["pomp2"]["l_rate"], 0.0],
                dt["hour"], state_per[j])

        elif components[j] == [1, 1, 1, 1]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"][r_rate_FE1],
data.lam["pomp2"]["r_rate"], data.lam["FE2"][r_rate_FE2]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"][r_rate_FE1],
data.r["pomp2"]["r_rate"], data.r["FE2"][r_rate_FE2]],
                dt["hour"], state_per[j])

        elif components[j] == [0, 1, 1, 0]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["l_rate"], 0.0, data.lam["pomp2"]["r_rate"],
data.lam["FE2"]["l_rate"]],
                [data.r["pomp1"]["l_rate"], 0.0, data.r["pomp2"]["r_rate"],
data.r["FE2"]["l_rate"]],
                dt["hour"], state_per[j])

        elif components[j] == [1, 0, 1, 1]:
            result = function.Determination_states(
                components[j], weight[j],
```

```
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"]["l_rate"],
data.lam["pomp2"]["r_rate"], data.lam["FE2"][r_rate_FE2]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"]["l_rate"],
data.r["pomp2"]["r_rate"], data.r["FE2"][r_rate_FE2]],
                dt["hour"], state_per[j])

        elif components[j] == [1, 1, 1, 0]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"][r_rate_FE1],
data.lam["pomp2"]["r_rate"], data.lam["FE2"]["l_rate"]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"][r_rate_FE1],
data.r["pomp2"]["r_rate"], data.r["FE2"]["l_rate"]],
                dt["hour"], state_per[j])

        elif components[j] == [1, 0, 1, 0]:
            result = function.Determination_states(
                components[j], weight[j],
                [data.lam["pomp1"]["r_rate"], data.lam["FE1"]["l_rate"],
data.lam["pomp2"]["r_rate"], data.lam["FE2"]["l_rate"]],
                [data.r["pomp1"]["r_rate"], data.r["FE1"]["l_rate"],
data.r["pomp2"]["r_rate"], data.r["FE2"]["l_rate"]],
                dt["hour"], state_per[j])

        components.append(result[0])
        state_base.append(result[1])
        state_per.append(result[2])
        weight.append(result[3])

#SFP1
        flow_rate_SFP1 = function.Flow_rate(components[j][0], components[j][1],
data.f_rate["pomp1"], data.f_rate["FE1"], water_level_SFP1[j], dt["hour"])[0]
        water_level_SFP1.append(function.Flow_rate(components[j][0], components[j][1],
data.f_rate["pomp1"], data.f_rate["FE1"], water_level_SFP1[j], dt["hour"])[1])
        water_weight_SFP1 = data.area * water_level_SFP1[j+1] * 10**3
        u_SFP1 += function.U(flow_rate_SFP1) - function.Qi(dt["second"])
```

```
        if u_SFP1 > 0:
            temp_SFP1.append(
                function.Temperature_avg(temp_SFP1[j], water_level_SFP1[j], flow_rate_SFP1)
                + function.Temperature_chg(dt["second"], water_weight_SFP1))
        else:
            temp_SFP1.append(100.0)
            water_level_SFP1[j+1] = max(0.0, water_level_SFP1[j+1] -
function.Evaporation(u_SFP1) * 10**(-3) / data.area)
            u_SFP1 = 0.0
        if water_level_SFP1[j+1] < data.length["TAF"]:
            append_cd_SFP1 = 1
        cd_SFP1.append(append_cd_SFP1)


#SFP2
        flow_rate_SFP2 = function.Flow_rate(components[j][2], components[j][3],
data.f_rate["pomp2"], data.f_rate["FE2"], water_level_SFP2[j], dt["hour"])[0]
        water_level_SFP2.append(function.Flow_rate(components[j][2], components[j][3],
data.f_rate["pomp2"], data.f_rate["FE2"], water_level_SFP2[j], dt["hour"])[1])
        water_weight_SFP2 = data.area * water_level_SFP2[j+1] * 10**3
        u_SFP2 += function.U(flow_rate_SFP2) - function.Qi(dt["second"])
        if u_SFP2 > 0:
            temp_SFP2.append(
                function.Temperature_avg(temp_SFP2[j], water_level_SFP2[j], flow_rate_SFP2)
                + function.Temperature_chg(dt["second"], water_weight_SFP2))
        else:
            temp_SFP2.append(100.0)
            water_level_SFP2[j+1] = max(0.0, water_level_SFP2[j+1] -
function.Evaporation(u_SFP2) * 10**(-3) / data.area)
            u_SFP2 = 0.0
        if water_level_SFP2[j+1] < data.length["TAF"]:
            append_cd_SFP2 = 1
        cd_SFP2.append(append_cd_SFP2)


    return time["hour"], temp_SFP1, temp_SFP2, water_level_SFP1, water_level_SFP2, state_base,
state_per, weight, cd_SFP1, cd_SFP2
```

```
#一つのサンプルの結果を表示
result = SFP_model(12001)

def Temperature_plt():
    plt.plot(result[0], result[1], color="red")
    plt.plot(result[0], result[2], color="blue")
    plt.xlabel("Time [h]")
    plt.ylabel("Temparature [℃]")
    plt.show()

def Water_level_plt():
    plt.ylim(-3, 14)
    plt.plot(result[0], result[3], color="red")
    plt.plot(result[0], result[4], color="blue")
    plt.xlabel("Time [h]")
    plt.ylabel("Water level [m]")
    plt.show()

def Temp_and_WL_plt():
    fig, ax1 = plt.subplots()
    ax1.plot(result[0], result[1], color="red")
    ax1.set_xlabel("Time [h]")
    ax1.set_ylabel("Temparature [℃]")
    ax2 = ax1.twinx()
    ax2.plot(result[0], result[2], color="blue")
    ax2.set_ylabel("water_level [m]")
    ax2.set_ylim(-3, 14)
    ax3 = ax1.twinx()
    ax3.plot(result[0], result[3], color="green")
    ax3.set_ylabel("state [-]")
    #plt.tight_layout()
    #for x,i in enumerate(state):
    #    if x != 0:
    #        ax2.axvspan(th[i],th[i+1], color= "coral")
```

```
    plt.show()


def U_plt():
    plt.plot(result[0], result[4], color="green")
    plt.xlabel("Time [h]")
    plt.ylabel("U []")
    plt.show()


def state_base_plt():
    plt.plot(result[0],result[5],color="green",label="State 0_CMMC")
    plt.xlabel("Time [h]")
    plt.ylabel("Probability [-]")
    plt.show()


#Temperature_plt()
#Water_level_plt()
#state_base_plt()
#Temp_and_WL_plt()
#U_plt()
```