2024 年度
応用プログラミング実験

第 6-8 回
確率プログラミング
実験レポート

学修番号: 22140026
谷 知拓 - Tomohiro Tani[*1]

第 1 回レポート提出日 : 2024-05-22
第 2 回レポート提出日 : 2024-05-30
第 2 回レポート提出日 : 2024-06-06

[*1] 東京都立大学 システムデザイン学部 情報科学科
mail@taniii.com

はじめに

本レポートでは,『応用プログラミング実験』第 6-8 回の実施報告を行う.

実験の概要

本実験では，確率プログラミングについて学ぶ．以下の課題に取り組み，その結果を報告する．

1. 課題 1-1: 逆関数法による乱数生成
2. 課題 1-2: 逆関数法による乱数生成
3. 課題 1-3: ランダムウォーク
4. 課題 1-A: 線形合同法

実験環境

実験環境は以下の通りである.

- OS[*2]: macOS Ventura 13.4.1
- CPU[*3]: Apple M2 arm64[*4]
- メインメモリ・ビデオメモリ共通: 16GB ユニファイドメモリ[*4]
- 実行環境 (rustc): rustc 1.78.0 (9b00956e5 2024-04-29)
- 実行環境 (cargo): cargo 1.78.0 (54d8815d0 2024-03-26)

[*2] Operating System
[*3] Central Processing Unit
[*4] https://www.apple.com/jp/macbook-air-13-and-15-m2/specs/

課題 1-1: 逆関数法による乱数生成

kadai_1_1_tomohiro_tani.rs を参照のこと.

なお，プログラムを実行して確認しやすくするために，上記プログラムと合わせて，依存関係の定義など実行に必要なファイルを含めたプロジェクトリポジトリ rust を添付している.

実行方法は以下の通りである.

まず，rust の実行環境は以下のコマンドでインストールすることができる.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

各課題は, works モジュール内のサブモジュールに含まれているので, それぞれのサブモジュールの main 関数を呼び出すことで実行できる.

各課題を実行するためには, src ディレクトリ直下の main 関数内で呼び出す関数を変更する.

実行は，プロジェクトリポジトリ直下で，以下のコマンドで行うことでできる.

```
cargo run
```

課題 1-2: 逆関数法による乱数生成

kadai_1_2_tomohiro_tani.rs を参照のこと.

概要

課題 1-1 で作成した関数 rnd_exp が，λ の指数分布に従う乱数を生成しているかを検証する．検証方法として，以下の 2 つの観点から理論値と比較する：

1. 乱数の平均値と分散
2. 乱数の分布

平均値と分散の検証

まず，$\lambda = 1.0, 1.5, 2.0$ の 3 つの異なるパラメータに対して，$n = 10,000$ 個の乱数を生成した．生成された乱数の平均値と分散を計算し，理論値と比較した.

理論値は以下の通り：

$$\text{平均値} : E[X] = \frac{1}{\lambda}$$
$$\text{分散} : V[X] = \frac{1}{\lambda^2}$$

シミュレーション結果と理論値の平均値及び分散を表 1にまとめる.

表 1　シミュレーション結果と理論値の比較

λ	平均値 (理論値)	平均値 (シミュレーション)	分散 (理論値)	分散 (シミュレーション)
1.0	1.000	0.998	1.000	1.018
1.5	0.667	0.658	0.444	0.452
2.0	0.500	0.493	0.250	0.240

シミュレーション結果は理論値と高い精度で一致しており，rnd_exp 関数が正しく指数分布に従う乱数を生成していることが確認できた.

分布の検証

次に, 生成された乱数の分布をプロットし, 理論的な確率密度関数と比較した．図 1 〜 図 3 は, $\lambda = 1.0, 1.5, 2.0$ に対する確率密度関数と生成された乱数のヒストグラムを示している.

これらの図から，生成された乱数が理論的な指数分布に高い精度で一致していることが確認できる.

考察

結果より，以下のように考察できる.

- 平均値と分散が理論値と高い精度で一致していることから，rnd_exp 関数が正しく動作していることが確認できた.

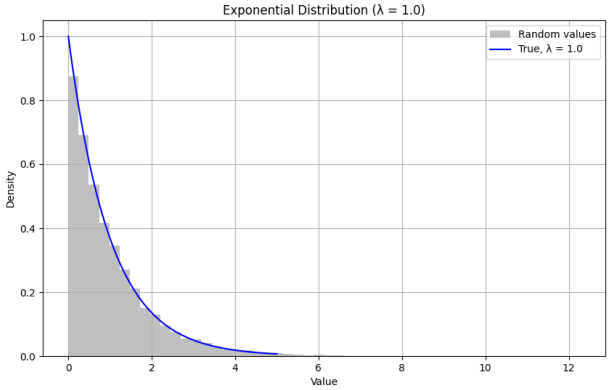

図 1　$\lambda = 1.0$ の指数分布

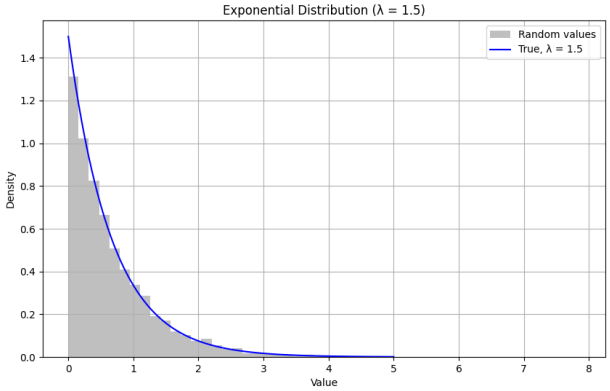

図 2　$\lambda = 1.5$ の指数分布

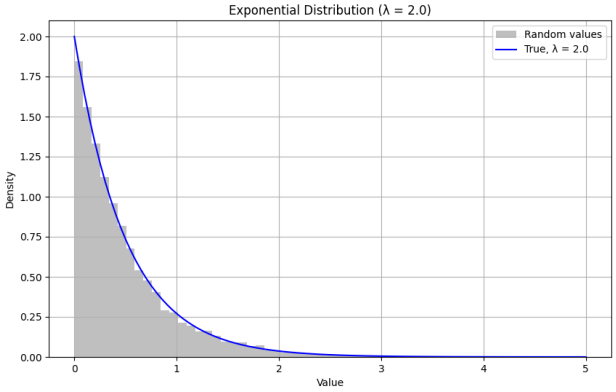

図 3　$\lambda = 2.0$ の指数分布

- 確率密度関数と乱数の分布が高い精度で一致していることから，生成された乱数が期待通りの分布に従っていることが確認できた.
- 逆関数法により，期待する確率分布に従う乱数を生成することができることが確認できた.

課題 1-3: ランダムウォーク

`kadai_1_3_tomohiro_tani.rs` を参照のこと.

概要

ランダムウォークのシミュレーションを行い，その結果を分析する．具体的には，$p = 0.5, d = 1$ の条件で 1 回のランダムウォーク（1000 ステップ）の軌跡をプロットし，100 回の独立なシミュレーションで得られる $t = 1000$ における点の位置 S_{1000} の平均値と分散を計算する．

結果

1 回のランダムウォークの軌跡

図 4 は，1 回のランダムウォークで得られた点の位置 $S_t(t = 0, 1, ..., 1000)$ の軌跡を示している．

図 4　ランダムウォークの軌跡

S_{1000} の平均値と分散

100 回の独立なシミュレーションを行い，$t = 1000$ における点の位置 S_{1000} の平均値と分散を計算した．
結果：

- 平均値： -1.080
- 分散： 1067.954

考察

- **平均値について**: 理論的には，ランダムウォークの各ステップが独立であり，正負の方向に等確率で進むため，長期的な期待値は 0 となる．シミュレーション結果の平均値も 0 の近傍にあることから，矛盾しない．

- **分散について**: 理論的には，ランダムウォークの分散はステップ数に比例し，分散は $t \cdot d^2$ となる．今回のシミュレーションでは，$d = 1$ であり，$t = 1000$ のとき，分散は理論的には 1000 となる．シミュレーション結果の分散もこれに非常に近い値であるため，理論的な予測が正しいことが確認できた．ただし，d をステップ毎の移動距離とする．
- **シミュレーションの信頼性について**: ランダムウォークのシミュレーション結果は理論的な期待値および分散と一致しており，このシミュレーション手法が信頼できるものであることがわかる．これにより，ランダムウォークを用いた複雑な現象のモデル化や解析が信頼性を持って行えることが示された．

これらの考察から，シミュレーションが理論的な予測と一致しており，乱数生成の品質が高いことが確認できた．ランダムウォークのモデルは，様々な分野における確率的現象の解析に有用であり，実験結果はその有効性を示しているといえる．

課題 2-1: ポアソン過程のイベントクラスの作成

kadai_2_1_tomohiro_tani.rs を参照のこと.

課題 2-2: ポアソン過程に従うイベント列を返す関数の作成

kadai_2_2_tomohiro_tani.rs を参照のこと.

課題 2-3: ポアソン過程のシミュレーション

`kadai_2_3_tomohiro_tani.rs` を参照のこと．

概要

課題 2-2 で作成した関数を用いて，ポアソン過程のシミュレーションを行う．具体的には，$\lambda = 1.0, 1.5, 2.0$ のポアソン過程に従う $n = 100,000$ のイベント列を生成し，時間区間 $T = 1$ のイベント発生回数 k の確率分布を考察する．

結果

図 5 〜 図 7 は，それぞれ $\lambda = 1.0, 1.5, 2.0$ のポアソン過程に従うイベント列を生成した結果である．
ただし，折れ線は理論値，点はシミュレーション結果を示している．

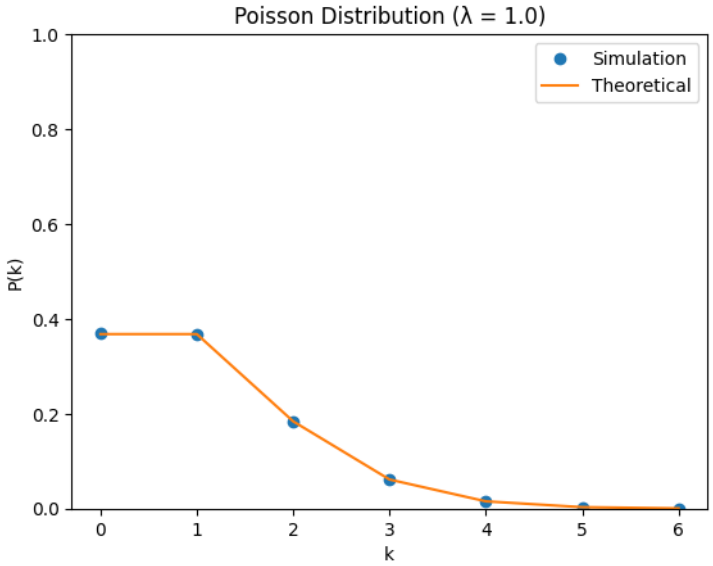

図 5　時間区間 $T = 1$ のイベント発生回数 k の確率分布 ($\lambda = 1.0$)

考察

結果より，以下のように考察できる．
ただし，ポアソン過程の理論値は以下の式の通りである．

$$P(X = k) = \frac{e^{-\lambda} \cdot \lambda^k}{k!} \tag{1}$$

- **ポアソン過程の特性について**: ポアソン過程は，一定の時間区間内でのイベント発生回数がポアソン分布に従う確率過程である．シミュレーション結果は理論値と一致しており，ポアソン過程の特性が正しく再現されていることが確認できた．
- **パラメータ λ について**: λ が大きいほど，イベント発生回数の平均値が大きくなることが確認できた．また，λ が大きいほど，イベント発生回数の分散も大きくなることが確認できた．これは，ポアソン過

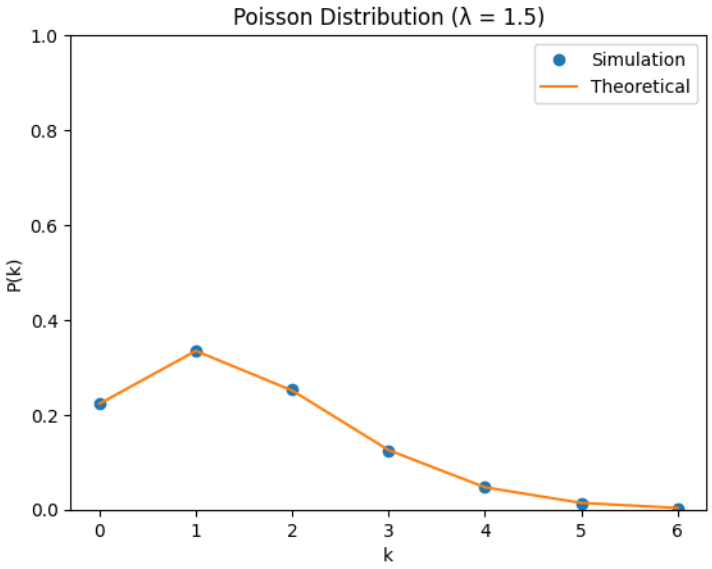

図 6　時間区間 $T = 1$ のイベント発生回数 k の確率分布 ($\lambda = 1.5$)

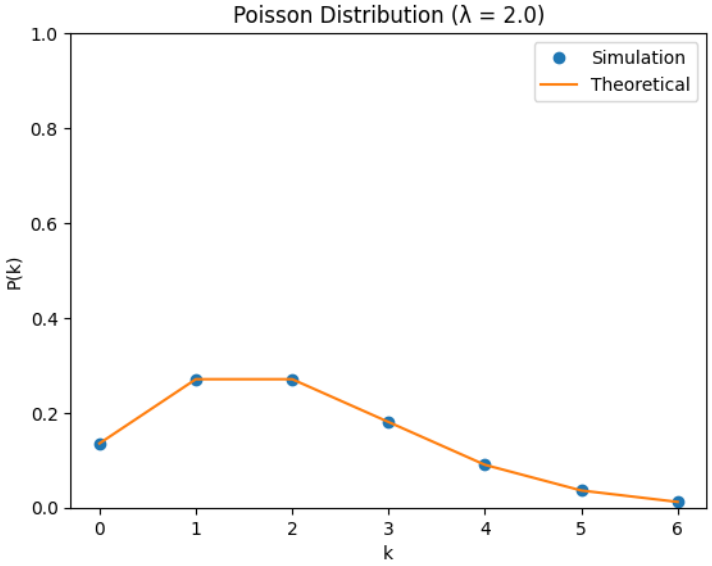

図 7　時間区間 $T = 1$ のイベント発生回数 k の確率分布 ($\lambda = 2.0$)

程の特性によるものであり，シミュレーション結果は理論的な予測と矛盾していないことがいえる．

課題 3-1: M/M/S/S シミュレータの作成

kadai_3_1_tomohiro_tani.rs を参照のこと.

課題 3-2: M/M/1/1 シミュレーション

`kadai_3_2_tomohiro_tani.rs` を参照のこと.

概要

課題 3-1 で作成した M/M/S/S シミュレータを用いて，M/M/1/1 シミュレーションを行う.
具体的には，$\lambda = 2.0$ のときの，$\mu = 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5$ のロス率について求める.

結果

図 8 は，$\lambda = 2.0$ のときの，$\mu = 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5$ に対するロス率をグラフにしたものである.

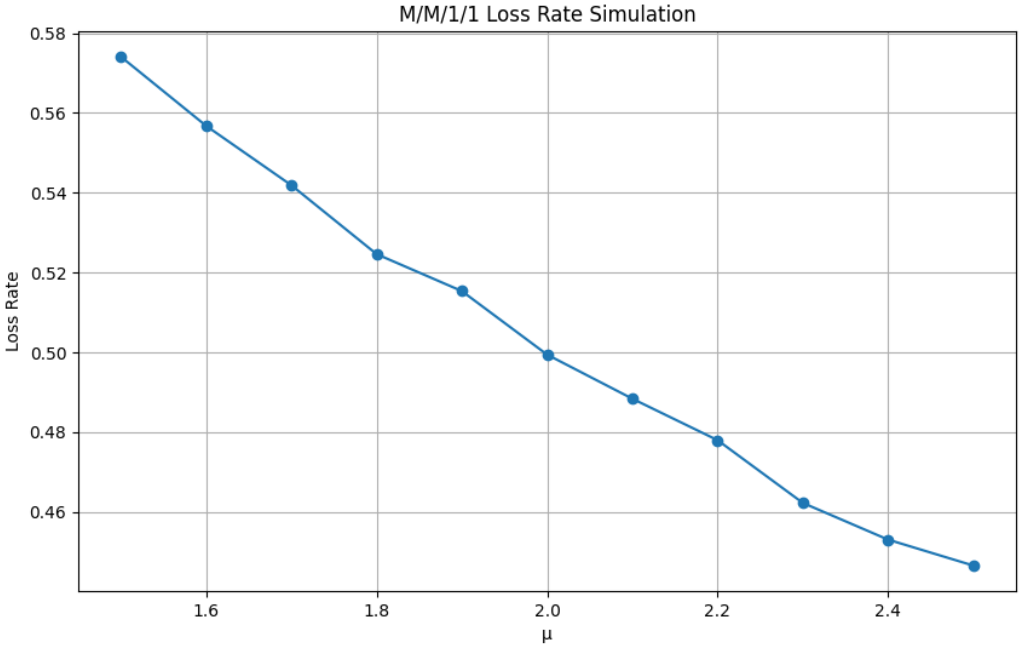

図 8　$\lambda = 2.0$ のときの，μ とロス率の関係

考察

このシミュレーションの理論値は以下の通りである.

$$\text{理論値のロス率} = \frac{\lambda}{\lambda + \mu} \tag{2}$$

図 9 は，理論値のグラフである.

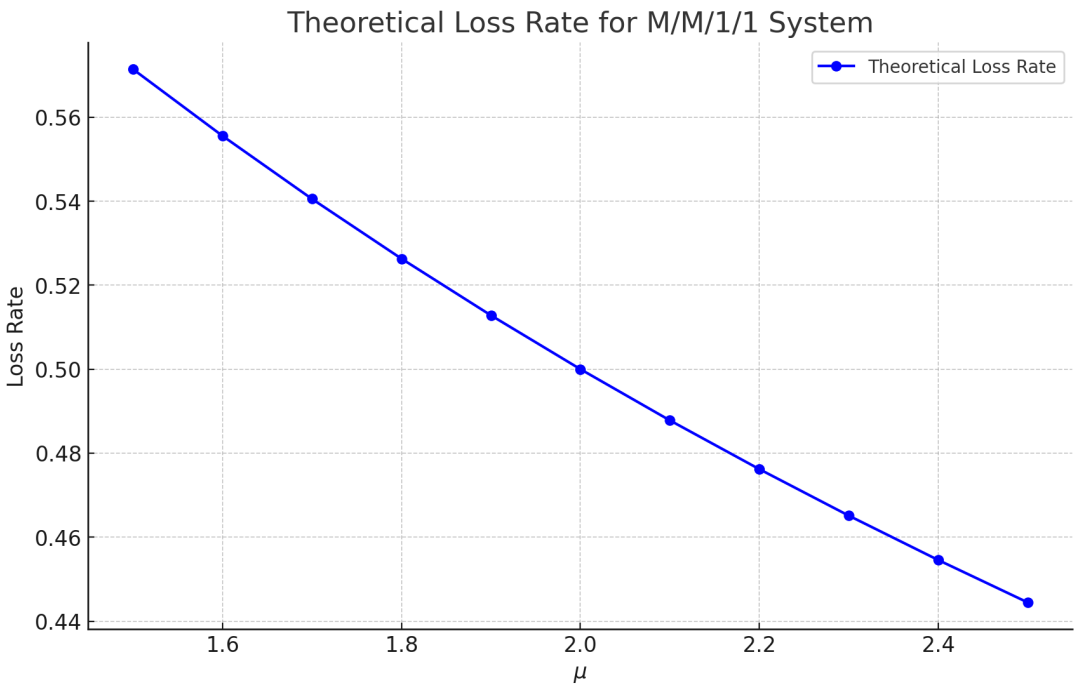

図 9　$\lambda = 2.0$ のときの，μ とロス率の関係 (理論値)

図 8 より，μ が大きいほど，ロス率が小さくなることが確認できる．これは，サービスの提供時間は $1/\mu$ に従う指数分布となるためであるといえる．

また，図 8，図 9 より，シミュレーション結果と理論値とは矛盾せず，正しいシミュレーションが行えたといえる．

課題 3-3: M/M/S/S シミュレーション

`kadai_3_3_tomohiro_tani.rs` を参照のこと.

概要

課題 3-1 で作成した M/M/S/S シミュレータを用いて，M/M/S/S 待ち行列をモデル化したときのロス率のシミュレーションを行う.

結果

図 10 は，$S = 1, 2, 3, 4, 5$ それぞれについて，$\lambda = 2.0$ とし，$\mu = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5$ に対するロス率をグラフにしたものである.

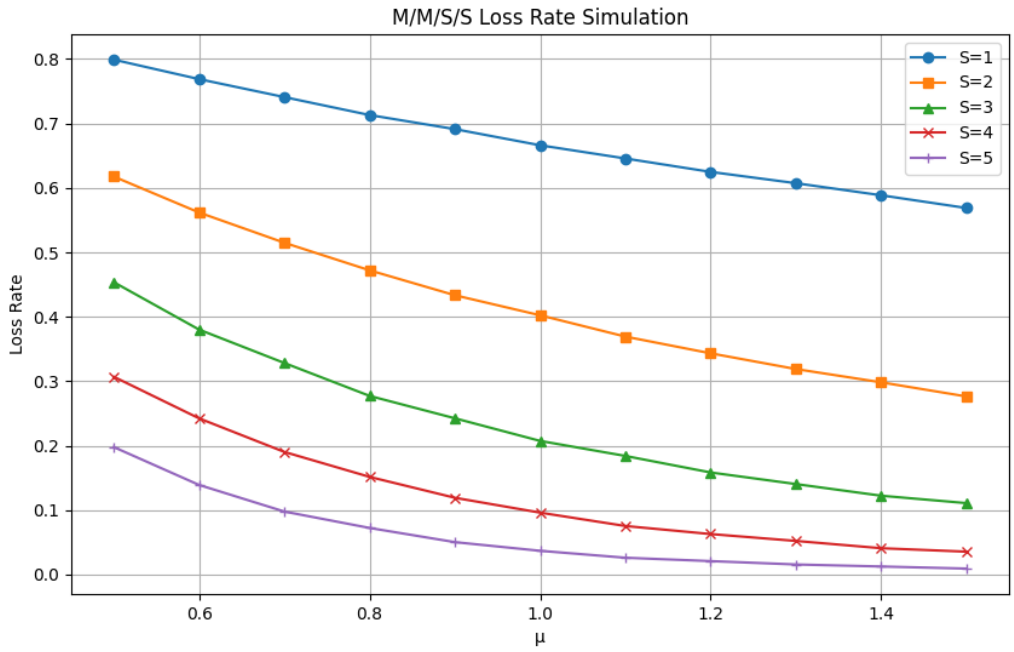

図 10　$S = 1, 2, 3, 4, 5$ それぞれについて，$\lambda = 2.0$ とし，μ とロス率の関係

考察

図 10 より，S が大きいほど，ロス率が小さくなることが確認できる．これは，S がサーバの数を表すためである.

サーバの数が多いほど，リクエストを処理するキャパシティが大きくなるため，ロス率が小さくなることが理解できる.

課題 3-A: M/M/1/∞ シミュレーション

`kadai_3_4_tomohiro_tani.rs` を参照のこと.

概要

課題 3-1 で作成した M/M/S/S シミュレータでは，サーバがすべて埋まっている場合にロスが発生する.

ここでは，サーバがすべて埋まっている場合に，新たなリクエストが到着した場合に，任意のサーバに空きが生じるまで待機する M/M/1/∞ シミュレータを作成し，シミュレーションする.

シミュレーションでは, $\lambda = 1.0, 1.5, 2.0, 2.5, 3.0$ のそれぞれのとき, $\mu = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5$ に対する待ち時間を求める.

結果

図 11 と表 2, 3 は, $\lambda = 1.0, 1.5, 2.0, 2.5, 3.0$ のそれぞれについて, $\mu = 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5$ に対する待ち時間を試行したものである.

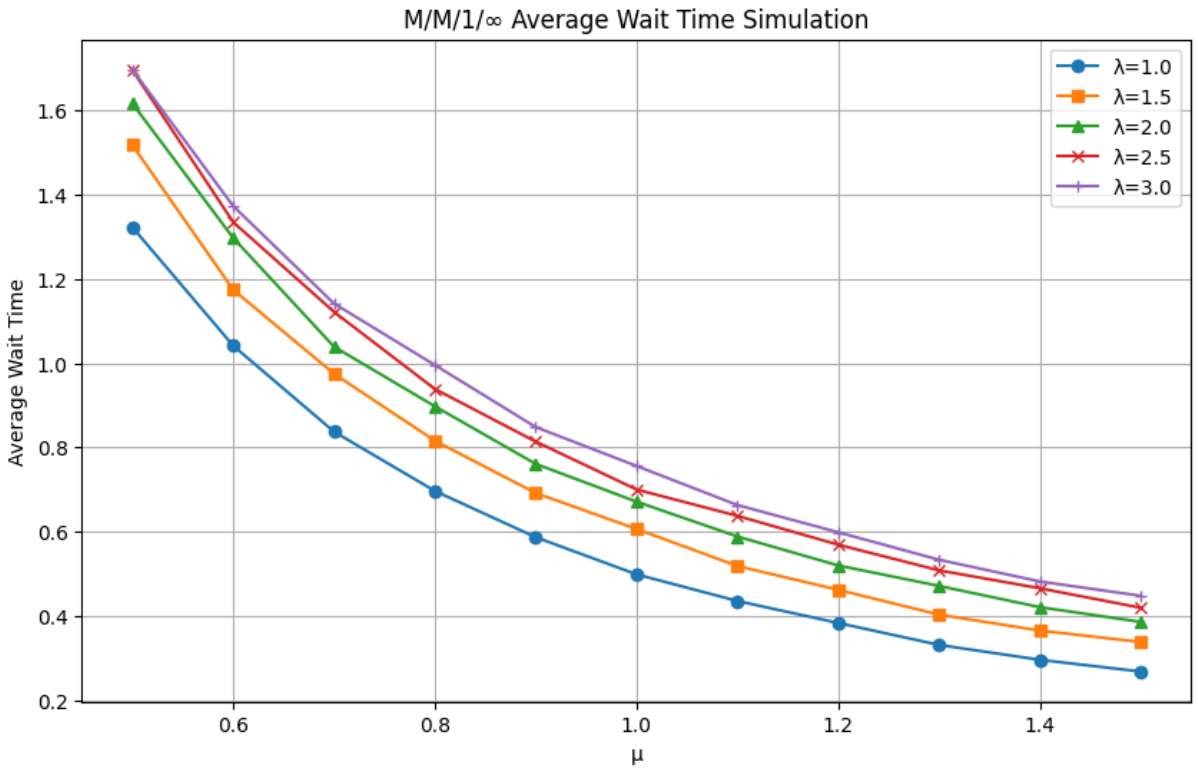

図 11　$\lambda = 1.0, 1.5, 2.0, 2.5, 3.0$ それぞれについて，μ と待ち時間の関係

考察

このシミュレーションの理論値は以下の通りである.

$$\text{理論値の平均待ち時間} = \frac{\lambda}{\mu(\mu - \lambda)} \tag{3}$$

図 11 より，μ が大きいほど待ち時間が短く，λ が大きいほど待ち時間が長くなることが確認できる．

これは，$\rho = \frac{\lambda}{\mu}$ が混み具合を表すことからも理解できる．

リクエストの到着率が大きく，サーバの処理時間 $1/\mu$ が長いほど，サーバがすべて埋まっている状態が続くため，待ち時間が大きくなることが理解できる．

おわりに

第 1 回では，様々なシミュレーションをコンピュータを用いて行うには，期待する分布に従った乱数生成の手法が必要不可欠であることがわかった．

また，乱数生成の品質がシミュレーション結果に大きな影響を与えるということを理解することができた．

第 2 回では，生成したイベントの発生回数の確率分布が，理論値とくるいなく一致しているのが印象的であった．

世界のさまざまな事象をモデル化するのに，確率分布という概念が非常に重要であることがわかった．

第 3 回では，待ち行列のモンテカルロシミュレーションを行うことができ，特に Queue を用いた非同期処理システムをモデル化できたのがとても興味深かった．

現在，開発業で，分散型メールシステムに関わる非同期処理システムの構築を行なっているため，今回の課題は非常に参考になった．

具体的には，メールサーバからの取得プロトコルである，IMAP は 40 年近く前の 1986 年に誕生したため，現代の Webhook のようなステートレスにイベントを受け取る仕組みが十分でなく，最新のメールが来ていないかどうか，メールアカウントごとに確認しにいく必要があり，ここに非同期の待ち行列モデルが生じるため，設計に試行錯誤している．

参考文献

- Learn Rust - Rust Programming Language: `https://www.rust-lang.org/learn` (accessed: May 22, 2024)
- Kleinrock, Leonard (1975). ”Queueing Systems Volume 1: Theory”

λ	μ	Average Wait Time
1	0.5	1.3228152698922535
1	0.6	1.041569830926313
1	0.7	0.8374044732034905
1	0.8	0.6967196489375518
1	0.9	0.5869698156477449
1	1	0.49875799140777366
1	1.1	0.43558179996194596
1	1.2	0.3833763555373002
1	1.3	0.33130003892426846
1	1.4	0.29597590419434566
1	1.5	0.2681681851815683
1.5	0.5	1.5179910691111629
1.5	0.6	1.1743131673277443
1.5	0.7	0.9748517752954529
1.5	0.8	0.8159103185325141
1.5	0.9	0.691728413085764
1.5	1	0.6064827751611467
1.5	1.1	0.5185512594446519
1.5	1.2	0.46209718876935896
1.5	1.3	0.40311204183920757
1.5	1.4	0.3653389920867382
1.5	1.5	0.3382454144182695

表 2　Average Wait Times for Different λ and μ Values (Part 1)

λ	μ	Average Wait Time
2	0.5	1.616443175247881
2	0.6	1.297284139505476
2	0.7	1.0390885645160224
2	0.8	0.8976514503134069
2	0.9	0.7609746883092575
2	1	0.6712300283393365
2	1.1	0.5886159523096386
2	1.2	0.5198381281538412
2	1.3	0.4713631072399001
2	1.4	0.4208963675680898
2	1.5	0.38592673299351204
2.5	0.5	1.6960399338643928
2.5	0.6	1.3348448122750503
2.5	0.7	1.1217175905202426
2.5	0.8	0.9387070482251122
2.5	0.9	0.8138929675559344
2.5	1	0.7001063316441805
2.5	1.1	0.6375614821933724
2.5	1.2	0.5691710428180768
2.5	1.3	0.5082233710422845
2.5	1.4	0.4656886944839204
2.5	1.5	0.41945248080787484
3	0.5	1.696860672213055
3	0.6	1.3730511708451394
3	0.7	1.1415333809844066
3	0.8	0.9955925998171699
3	0.9	0.8482943727529185
3	1	0.7559943425955813
3	1.1	0.6632060206479782
3	1.2	0.5985737201894004
3	1.3	0.5333848377164928
3	1.4	0.48189329603655956
3	1.5	0.4479137556725958

表 3　Average Wait Times for Different λ and μ Values (Part 2)