信州大学大学院

修士論文

情報鮮度の視点から見たサーバで割り込みのある待ち行列通信システムの解析

指導教員　西新 幹彦 准教授

工学専攻　電子情報システム工学分野
学籍番号　22W2093E
氏名　細海 俊介

2024 年 3 月 7 日

目次

1　はじめに

防犯カメラやネットワークカメラなどを使用して，遠く離れた場所に異常がないかどうか観察し，その映像をモニタに表示する場合を考える．カメラとモニタの場所は互いに離れていて，その間には通信路が設置されている．カメラで撮影した情報は通信路を通過する間に古くなり，モニタに表示される時点では，程度はともかく過去のものとなってしまう．さらに，表示された情報も次の情報が届くまで時間の経過とともに古くなり続ける．しかしシステムの目的からして，モニタに表示される情報はできるだけ新しいことが望ましい．このような情報の新しさは情報鮮度 (Age of Information: AoI) と呼ばれる．AoI が小さいほど情報が新しいことを意味する．

待ち行列理論の立場に基づく従来研究では，システムの設定と AoI の関係を数学的に解析することが行われている．例えば，研究 [1] ではセンサでサンプリングされた情報が送信機に到着した順番に処理される FIFO(First In First Out) のシステムの AoI について考えられている．また，別の研究 [2] では，送信機の処理中に新しい情報が到着したら古い情報の処理を中断し割り込みをして優先的に処理する LIFO（Last In First Out）を用いたシステムの AoI について考えられている．そして，LIFO の方が FIFO より AoI が小さくなることが明らかにされている．これらの研究で想定されたシステムでは，情報はパケットごと処理し，サービス時間は指数分布に従うものであった．

さらにその後，この問題は，設定の中に符号器が追加され，情報理論的な最適化問題として考えられるようになった．研究 [3] では情報を符号語に変換し，シミュレーション実験によって AoI の変化が調べられている．

本研究では，サーバで割り込みがある待ち行列通信システムを数理的に解析して，AoI の数学的表現を導いた．

本論文の以降の構成は以下の通りである．2 章では，AoI の定義，時間平均 AoI の計算方法について述べる．3 章では，符号器が追加されていないシステムに関する従来研究 [4] の紹介をする．4 章では，サービス時間が指数分布に従う符号器を追加した通信モデルの詳細について述べる．5 章では，AoI と割り込みが発生しどれだけの情報が棄却されたのかという棄却率の解析した値とシミュレーションの結果との差異など様々な視点からみた考察を示す．6 章では，本論文のまとめと今後の課題について述べる．

2　AoI とその評価方法

図 1 のような通信システムを考える．このシステムでは情報源から出力されたシンボルは到着順に送信機のバッファに格納され保管される．サーバではバッファに到着した順番でシンボ

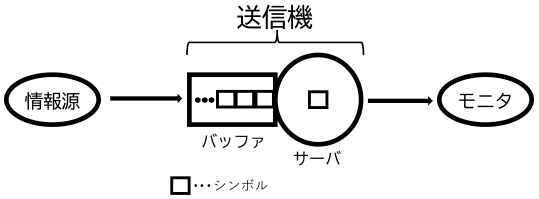

図 1　通信システムの例

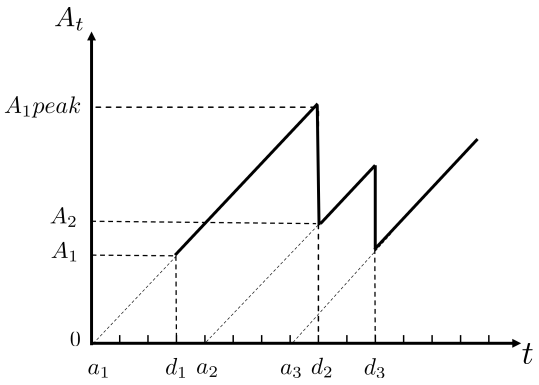

図 2　AoI の遷移図

ルの処理が行われ通信路を通り遠隔地のモニタに表示される．情報が発生してからの経過時間はその情報の年齢と呼ばれる．本研究ではモニタに表示されている情報の年齢を考察する．以降はこれを単に AoI と呼ぶ．AoI は，情報が発生した時点からの経過時間に比例して斜め 45 ° で増加していく．そしてモニタに新しい情報が届いたときその年齢まで瞬時に降下する．このように，AoI は時刻と共に変化する．時刻 t にモニタに表示されている情報の AoI A_t はその情報の発生時刻を a とおくと，

$$A_t = t - a \tag{1}$$

と書ける．図 2 に AoI の時間変化の例を示す．1 番目に発生したシンボル X_1 が時刻 a_1 でバッファに保持される．a_1 から時間が経過するとシンボル X_1 の AoI が増加していき情報は古くなる．時刻 d_1 にサーバの処理が終了するとモニタにシンボル X_1 の情報が表示される．シンボル X_1 が表示された瞬間の値は，

$$A_1 = d_1 - a_1 \tag{2}$$

と書ける．さらに，時刻 a_2 で 2 番目のシンボル X_2 が発生する．ここで時刻 d_2 に注目する．このとき，直前までモニタに表示されていた情報は更新されてシンボル X_2 がモニタに表示さ

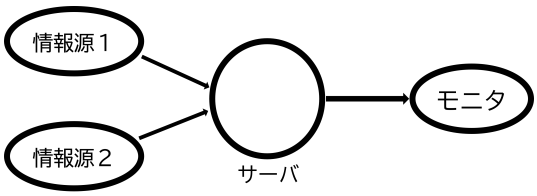

図 3　サーバで割り込みが発生する通信システム

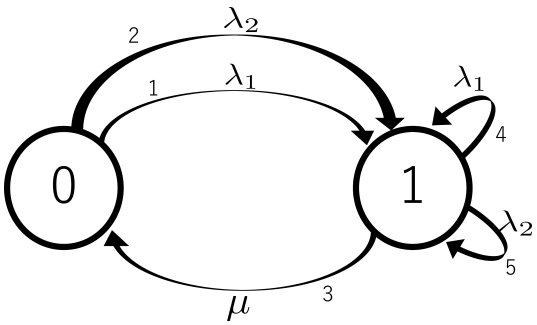

図 4　サーバで割り込みが発生する通信システムの状態遷移図

れる．モニタの表示内容が更新されることで，AoI の do 値が A_1peak から A_2 と小さくなる．このようにモニタの表示内容は更新を繰り返すことから，図 2 のノコギリ状の実線グラフが得られる．これが，モニタに表示された情報の AoI の時間的変化である．

AoI の評価方法には，ピーク AoI の平均を評価する方法や，AoI の時間平均を評価する方法などがある．本論文では評価方法として AoI の時間平均を評価する方法を採用する．時刻 t における AoI を A_t と表すと，本論文で評価する AoI の時間平均は

$$A_{ave} = \lim_{T\to\infty} \frac{1}{T} \int_0^T A_t dt \tag{3}$$

と書ける．

3　従来研究の紹介

シンボルがモニタに到着する間隔が小さいほど AoI の値が小さくなりより新しい情報が表示されることになる．そして，情報がモニタに表示された時点での AoI はバッファでの待ち時間とサーバでのサービス時間によって決まる．そこで，それらの時間を短くするために古いシンボルを破棄して新しいシンボルを優先的に処理する方法が考えられる．以下でそのようなシステムに対する従来研究 [5] の紹介をする．

表 1　図 5 のマルコフ連鎖の遷移表

l	$q_l \to q_l'$	$\lambda^{(l)}$	$\boldsymbol{x}\boldsymbol{A}_l$	$\boldsymbol{A}_l$	$\boldsymbol{v}_{q_l}\boldsymbol{A}_l$
1	0→1	λ_1	$[x_0 \ \ 0\,]$	$\left[\begin{smallmatrix}1 & 0\\ 0 & 0\end{smallmatrix}\right]$	$[v_{00} \ \ 0\,]$
2	0→1	λ_2	$[x_0 \ x_0\,]$	$\left[\begin{smallmatrix}1 & 1\\ 0 & 0\end{smallmatrix}\right]$	$[v_{00} \ v_{00}]$
3	1→0	μ	$[x_1 \ \ *\,]$	$\left[\begin{smallmatrix}0 & *\\ 1 & *\end{smallmatrix}\right]$	$[v_{11} \ \ *\,]$
4	1→1	λ_1	$[x_0 \ \ 0\,]$	$\left[\begin{smallmatrix}1 & 0\\ 0 & 0\end{smallmatrix}\right]$	$[v_{10} \ \ 0\,]$
5	1→1	λ_2	$[x_0 \ x_0\,]$	$\left[\begin{smallmatrix}1 & 1\\ 0 & 0\end{smallmatrix}\right]$	$[v_{10} \ v_{10}]$

3.1　サーバで割り込みが発生するシステム

サーバで割り込みが発生する通信システムを図 3 に示す．情報源は 2 種類あり，AoI を調べたいシンボルが発生する情報源を情報源 1，それ以外のシンボルが発生する情報源をすべてあわせて情報源 2 とみなしそれぞれ情報がモニタに映される．シンボルはポアソン到着すると考え，情報源 1 からは到着レート λ_1 で出力され，情報源 2 からは到着レート λ_2 で出力される．サーバのサービス時間は指数分布に従う．よって，情報源から出力されたシンボルは送信機にあるサーバに入れられサービスレート μ で処理され遠隔地のモニタにシンボルが表示される．ここで，サーバで処理中に新しいシンボルがサーバに到着すると処理中シンボルを棄却し新しいシンボルを処理することでモニタに表示されているシンボルの AoI を小さくすることができる．サーバの状態は空の状態かパケットの処理中かのどちらかであり，次のパケットの到着や処理の終了によって状態が遷移する．この様子を表した状態遷移図を図 4 に示す．図中の状態 0 と状態 1 は，それぞれサーバが空の状態と処理中を表している．情報源 1 と 2 からそれぞれ到着レート λ_1 と λ_2 でシンボルが到着する．いずれかの情報源からシンボルが到着するとサーバは処理中の状態へ遷移する．サーバは処理を終えるとサーバが空の状態になる．ただしサーバの処理中にもいずれかの情報源から情報が届く可能性があり，その場合は処理中のシンボルを破棄して新しいシンボルの処理が改めて開始される．

このようなシステムの AoI を数学的に計算する方法が Yates ら [2] によって確立されている．計算方法の詳細は付録 B に掲載する．

AoI を求めるために，まずシステムの状態遷移の詳細を表 1 のように書き出す．

次にこれを元に

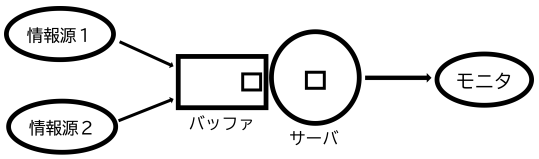

図５　バッファで割り込みが発生する通信システム

$$[v_{00}\ v_{01}](\lambda_1+\lambda_2) = [1\ 0]\bar{\pi}_0 + \mu[v_{11}\ *] \tag{4}$$
$$[v_{10}\ v_{11}](\mu+\lambda_1+\lambda_2) = [1\ 1]\bar{\pi}_1 + \lambda_1[v_{00}\ 0] + \lambda_2[v_{00}\ v_{00}] + \lambda_1[v_{10}\ 0] + \lambda 2[v_{10}\ v_{10}] \tag{5}$$
$$\bar{\pi}_0(\lambda_1+\lambda_2) = \mu\bar{\pi}_1 \tag{6}$$
$$\bar{\pi}_1(\mu+\lambda_1+\lambda_2) = \lambda_1\bar{\pi}_0 + \lambda_2\bar{\pi}_0 + \lambda_2\bar{\pi}_1 \tag{7}$$

という連立方程式を立てる．

これを $\bar{\pi}$ について解くとシステムの定常状態が分かり，時間平均 AoIΔ

$$\Delta = \frac{1+\rho}{\mu\rho_1} \tag{8}$$
$$= \frac{\mu+\lambda_1+\lambda_2}{\mu\lambda_1} \tag{9}$$

が得られる．このときの ρ とは

$$\rho = \frac{\lambda}{\mu} \tag{10}$$
$$\rho_1 = \frac{\lambda_1}{\mu} \tag{11}$$
$$\rho_2 = \frac{\lambda_2}{\mu} \tag{12}$$
$$\rho = \rho_1 + \rho_2 \tag{13}$$

で表される値であり，稼働率という．

3.2　バッファで割り込みが発生するシステム

バッファで割り込みが発生する通信システムを図 5 に示す．前節のシステムにバッファが追加されたシステムとなっている．サーバの前段にバッファが設置してありサーバが処理中の時に新しいシンボルが出力されたら 1 つだけ保持することができる．ここで，サーバで処理中かつバッファでシンボルが保持されているとき新しいシンボルがバッファに到着するとバッファ内のシンボルを棄却し新しいシンボルを保持することで，モニタに表示されているシンボ

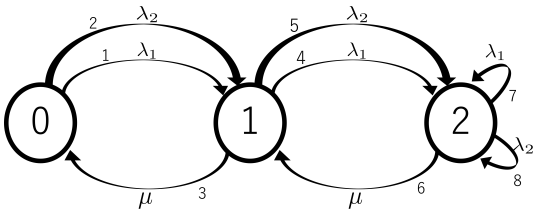

図 6　バッファで割り込みが発生する通信システムの状態遷移図

表 2　図 7 のマルコフ連鎖の遷移表

l	$q_l \to q_l'$	$\lambda^{(l)}$	$\boldsymbol{x}\boldsymbol{A}_l$	$\boldsymbol{v}_{ql}\boldsymbol{A}_l$
1	0→1	λ_1	$[x_0 \ \ 0 \ \ *\ ]$	$[v_{00} \ \ 0 \ \ *\ ]$
2	0→1	λ_2	$[x_0 \ x_0 \ *\ ]$	$[v_{00} \ v_{00} \ *\ ]$
3	1→0	μ	$[x_1 \ *\ \ *\ ]$	$[v_{11} \ *\ \ *\ ]$
4	1→2	λ_1	$[x_0 \ x_1 \ 0\ ]$	$[v_{10} \ v_{11} \ \ 0\ ]$
5	1→2	λ_2	$[x_0 \ x_1 \ x_1]$	$[v_{10} \ v_{11} \ v_{11}]$
6	2→1	μ	$[x_1 \ x_2 \ *\ ]$	$[v_{21} \ v_{22} \ *\ ]$
7	2→2	λ_1	$[x_0 \ x_1 \ 0\ ]$	$[v_{20} \ v_{21} \ \ 0\ ]$
8	2→2	λ_2	$[x_0 \ x_1 \ x_1]$	$[v_{20} \ v_{21} \ v_{21}]$

ルの AoI を小さくすることができる．状態遷移図を図 6 に示す．図中の状態 0 と状態 1 と状態 2 は，それぞれバッファが空でサーバも空，バッファが空でサーバ処理中，バッファ保持かつサーバ処理中を表している．各状態へ向かう遷移についてはサーバで割り込みが発生する場合と同様である．このシステムも前節と同じ手順で AoI を求められる．前節と同様にシステムの状態遷移の詳細を表 2 のように書き出す．

これをもとに

$$[v_{00}\ v_{01}\ v_{02}](\lambda_1+\lambda_2) = [1\ 0\ 0]\bar{\pi}_0 + \mu[v_{11}\ *\ *] \tag{14}$$

$$[v_{10}\ v_{11}\ v_{12}](\mu+\lambda_1+\lambda_2) = [1\ 1\ 0]\bar{\pi}_1 + \lambda_1[v_{00}\ 0\ *] + \lambda_2[v_{00}\ v_{00}\ *] + \mu[v_{21}\ v_{21}\ *] \tag{15}$$

$$[v_{20}\ v_{21}\ v_{22}](\mu+\lambda_1+\lambda_2) = [1\ 1\ 1]\bar{\pi}_2 + \lambda_1[v_{10}\ v_{11}\ 0] + \lambda_2[v_{10}\ v_{11}\ v_{11}] \tag{16}$$

$$+ \lambda_1[v_{20}\ v_{21}\ 0] + \lambda_2[v_{20}\ v_{21}\ v_{21}] \tag{17}$$

$$\bar{\pi}_0(\lambda_1+\lambda_2) = \mu\bar{\pi}_1 \tag{18}$$

$$\bar{\pi}_1(\mu+\lambda_1+\lambda_2) = \lambda_1\bar{\pi}_0 + \lambda_2\bar{\pi}_0 + \mu\bar{\pi}_2 \tag{19}$$

$$\bar{\pi}_2(\mu+\lambda_1+\lambda_2) = \lambda_1\bar{\pi}_1 + \lambda_2\bar{\pi}_1 + \lambda_1\bar{\pi}_2 + \lambda_2\bar{\pi}_2 \tag{20}$$

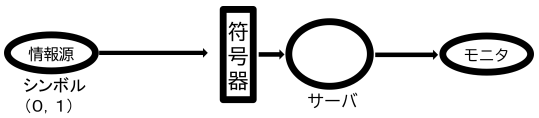

図 7　符号器を追加したサーバで割り込みが発生する通信システム

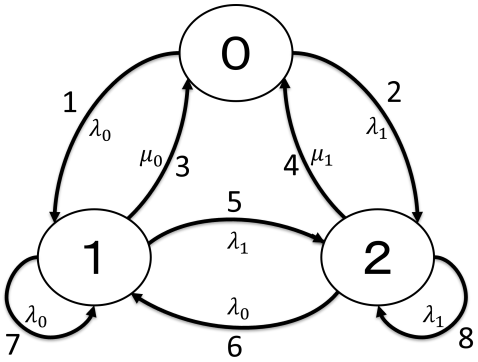

図 8　符号器を追加したサーバで割り込みが発生する通信システムの状態遷移図

という連立方程式を立てる．これを解くと時間平均 AoIΔ は

$$\Delta = \frac{1}{\mu} + \frac{\pi_0 + \pi_2}{\rho} + \frac{1}{\mu(1+\rho)}\left(\frac{1+\rho+\rho^2}{\rho_1} - \frac{1+\rho+\rho^3}{\rho(1+\rho)}\right) \tag{21}$$

と書ける．

4　サーバで割り込みが発生する符号器を追加した通信システム

3 章では，従来研究として 2 種類のシステムについて説明した．これらの研究ではどのシンボルも同じサービスレートで処理されていた．本研究ではシステムに符号器を追加しシンボルごとにサービスレートが変わる通信モデルを設定し，AoI の解析を行った．通信システムの構成は図 7 である．情報源のシンボルの種類は簡単のため ‘0’,‘1’ の 2 種類とする．シンボルは符号器にポアソン到着し，符号器で符号語に変換されサーバで処理され，遠隔地にあるモニタに映し出される．サーバでのサービス時間は符号語長を期待値とした指数分布によって決定する．符号語は語頭符号なのでクラフトの不等式を満たす．クラフトの不等式は付録 A に掲載する．通常，符号語長は整数でなければならないがサービス時間を実数で表現するため符号語長を実数として扱う．モニタでは符号語が復号され表示される．新しい情報を常に送りたいのでサーバでシンボルが処理されているときに新しいシンボルが到着した場合，処理しているシンボルを棄却し，新しいシンボルを処理する．考察するに当たって検討する内容は時間平均

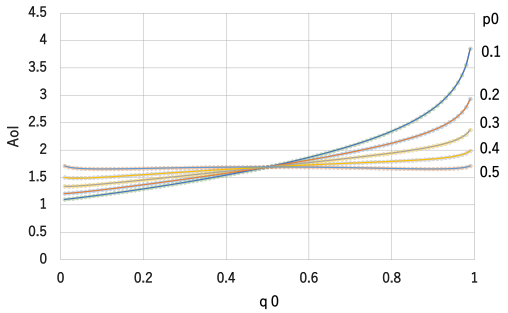

図 9　実験値と理論値を重ねた図 (AoI)

AoI，情報源から発生したシンボルの個数に対する棄却されたシンボルの個数の割合で定義される棄却率とした．時間平均 AoI は 3 章で説明した方法で導出する．棄却率を数学的に計算する方法が先行研究 [6] にて考えられている．計算方法の詳細は付録 A に掲載する．

AoIΔ，棄却率 ε を導出すると，

$$\Delta = \frac{1}{\lambda_0\mu_0 + \lambda_1\mu_1 + \mu_0\mu_1}\left(\lambda + \mu_0 + \mu_1 + \frac{\mu_0\mu_1}{\lambda}\right) \tag{22}$$

$$\varepsilon = 1 - \left(\frac{P_0}{1 - \log_e q_0} + \frac{P_1}{1 - \log_e (1 - q_0)}\right) \tag{23}$$

このような値が求められた．導出過程は付録 A，付録 B に記載する．このとき AoIΔ は λ が任意であるが棄却率 ε は λ=1.0 である．

5　結果と考察

5.1　シミュレーションの説明

数理的に求めた値と実験的に求めた値とを確認するためにプログラムで通信システムをシミュレートした．具体的には，サーバのサービス時間を指数分布に従わせ，パラメータとして情報源のシンボルの種類を ‘0’,‘1’，情報源からモニタに映し出された情報が 1000000 個になったら終了するようにした．

5.2　分布を変更したときの AoI

シンボル ‘0’ の発生確率 p_0 を 0.1 から 0.5 と変更したときの式 (22) とシミュレーション実験の値を重ねた図を図 9 に示す．横軸 q_0 は式 (30) で定まる値である．横軸の値が小さいほどシンボル ‘0’ の符号語長が長くなるためサーバでのサービス時間は大きくなる．図 9 の

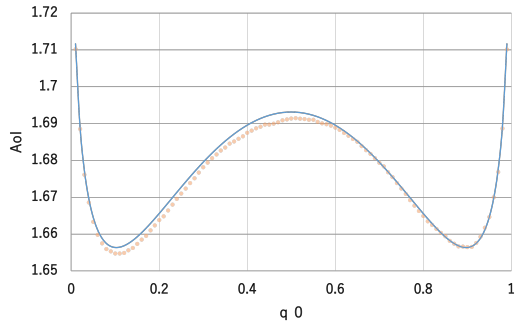

図 10　実験値と理論値を重ねた図 (AoI p_0=0.5)

縦軸は時間平均 AoI の値であり，縦軸の値が小さいほどモニタに表示される情報は新しいことになる．グラフで実線になっているのが式 (22) で点でプロットされているのがシミュレーション実験で得られた値である．グラフより，式 (22) は正しいことが分かる．このときの全体の到着レートは λ=1.0 である．まず，モニタに表示されている情報の AoI を小さくするためにはシンボル ‘0’,‘1’ のサービス時間を限りなく小さくすればよい．しかし，クラフトの不等式の制限があるため一方のサービス時間を短くするともう一方のサービス時間が大きくなる関係にある．各グラフの最小値は p_0=0.1 のとき (q_0,AoI)=(0.000901,1.1015)，p_0=0.2 のとき (q_0,AoI)=(0.00279,1.2227)，p_0=0.3 のとき (q_0,AoI)=(0.006268,1.3703)，p_0=0.4 のとき (q_0,AoI)=(0.012898,1.5529)，p_0=0.5 のときだけ最小値が (q_0,AoI)=(0.0275,1.7809)，(q_0,AoI)=(0.9725,1.7809) の 2 つある．ここで p_0=0.5 のみのグラフを図 10 に示す．このグラフからも最小値が 2 つあることが分かる．p_0 が小さくなるほど最小値はグラフの左下に向かう傾向が得られた．これは，情報源の分布が偏っているため頻繁に出力されるシンボルはサービス時間を短くし，稀に出力されるシンボルのサービス時間を大きくすることで全体としてモニタには新しい情報が表示され時間平均 AoI は小さくなるためである．符号器だけなくしたシステムを q_0=0.5 のときだと考えられる．図 9 より q_0 の値を変えると時間平均 AoI の最小値が q_0=0.5 のときより低くなること分かる．したがって，符号器を追加すると時間平均 AoI を小さくできると言える．

5.3　分布を変更したときの棄却率

前節では時間平均 AoI についての結果を示した．この節ではサーバで割り込みが発生した割合である棄却率について考える．まず，シンボル ‘0’ の発生確率 p_0 を 0.1 から 0.5 まで変更したときの式 (23) とシミュレーション実験の値を重ねた図を図 11 に示す．横軸は q_0，縦軸は棄却率であり，値が 0 に近づくほど割り込みが少なくなり情報源から出力されたシンボ

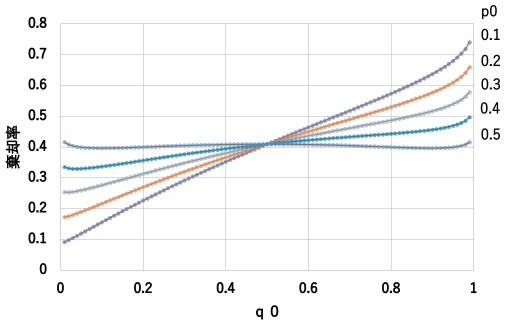

図 11　実験値と理論値を重ねた図 (棄却率)

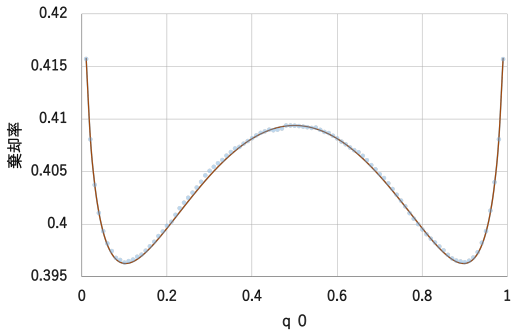

図 12　実験値と理論値を重ねた図 (棄却率 p_0=0.5)

ルが棄却されずにモニタに表示される．このとき棄却率はシンボルごとではなくシンボル全体の棄却率を計測した．これは，シンボルごとの棄却率を見てもそれはシンボルの発生確率に依存するため計測しても自明だからである．グラフで実線になっているのが式 (23) で点でプロットされているのがシミュレーション実験で得られた値である．グラフより，式 (23) も正しいことが分かる．このときの全体の到着レートは λ=1.0 である．評価する対象が変わってもクラフトの不等式の制限があるため，一方のサービス時間を短くするともう一方のサービス時間が大きくなる関係は変わらずに存在する．各グラフの最小値は p_0=0.1 のとき (q_0, 棄却率)=(0.002196,0.08793)，p_0=0.2 のとき (q_0, 棄却率)=(0.00713,0.17203)，p_0=0.3 のとき (q_0, 棄却率)=(0.01687,0.25268)，p_0=0.4 のとき (q_0, 棄却率)=(0.0379,0.3287)，AoI と同様に p_0=0.5 のときだけ最小値が (q_0, 棄却率)=(0.1027,0.3963)，(q_0, 棄却率)=(0.8973,0.3963) の 2 つある．ここで p_0=0.5 のみのグラフを図 12 に示す．このグラフからも最小値をとる点が 2 つあることが分かる．p_0 が小さくなるほど最小値はグラフの左下に向かう傾向が得られた．これは，情報源の分布が偏っているため頻繁に出力されるシンボルはサービス時間を短く

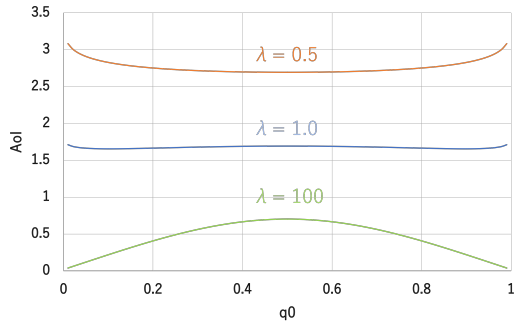

図 13　p_0=p_1=0.5 で λ を変更したときの AoI

し，稀に出力されるシンボルのサービス時間は大きくすることで全体として到着間隔よりもサービス時間が短くなり，棄却が発生しづらくなる．分布の偏りが大きいほどサービス時間の配分も偏り，頻繁に出力されるシンボルを短いサービス時間で処理することとなりグラフの最小値が右下に向かうと考えられる．また，前節と同様に符号器のないシステムを q_0=0.5 のときだと考えることができる．図 11 より q_0 の値を変えると棄却率の最小値が q_0=0.5 のときより低くなることが分かる．したがって，符号器を追加すると棄却率を小さくできると言える．

5.4　レートを変更したときの AoI

5.2 節ではシンボルの到着レート λ=1.0 のときの分布を変えた場合の AoI について考察した．このとき到着レート λ を変更したらどのように AoI が変化するのか気になった．比較しやすいように p_0=p_1=0.5 のとき λ を変更した．比較したグラフを図 13 に示す．横軸は q_0，縦軸は AoI である．λ=0.5 のとき下に凸のグラフ，λ=1.0 のとき図 10 のように波打つグラフ，λ=100 のときは上に凸のグラフが得られた．各グラフの最小値は λ=0.5 のとき (q_0,AoI)=(0.5,2.693)，λ=1.0 のとき (q_0, 棄却率)=(0.1027,0.3963)，(q_0, 棄却率)=(0.8973,0.3963) の 2 つ，λ=100 のときはグラフの両端の $q_0 \to 0$ と $q_0 \to 1$ のときである．これらよりレート λ を大きくすると AoI が小さくなることが分かった．

5.5　$\lambda \to \infty$ のときの AoI

前節で到着レート λ を大きくすると AoI が小さくなることが分かった．そこで，到着レート λ を ∞ にしたらどうなるのか知りたくなり式 (19) を $\lambda \to \infty$ で計算すると，

表 3　λ=1000,p_0=0.1 のときモニタ目線 1 秒あたりどれだけシンボルが届いているか

q_0	1 秒あたり届いたシンボル数	AoI
0.1	8.499920899	0.117532
0.2	4.073227472	0.245341
0.3	2.597302245	0.384799
0.4	1.867575782	0.53511
0.5	1.440358664	0.694153
0.6	1.176681395	0.849869
0.7	1.026646695	0.973662
0.8	1.004611283	0.994984
0.9	1.33129108	0.751491

$$\Delta = \frac{1}{\lambda_0\mu_0 + \lambda_1\mu_1 + \mu_0\mu_1}\left(\lambda + \mu_0 + \mu_1 + \frac{\mu_0\mu_1}{\lambda}\right) \tag{24}$$

$$\Delta = \frac{1}{p_0\mu_0 + p_1\mu_1} \quad (\lambda \to \infty) \tag{25}$$

となった．確認するためにシミュレーション実験でモニタから見て 1 秒間あたりにどれだけのシンボルが届いているのか検証した．λ=1000，p_0=0.1，q_0 を 0.1 から 0.9 まで変更して確認した．確認方法としてモニタに情報が 1000000 個届いたら終了するプログラムなので 1000000 を全体の時間で割ることで 1 秒間あたりの届いたシンボルを算出した。計算すると，

このような表を得た．これより λ が大きい値でもモニタにシンボルが届いていることが分かった．サーバで割り込みが発生するシステムでは到着レートを大きくしすぎると毎回棄却されモニタに情報が届かないように思えるがそうではないことが判明した．これはサーバが指数分布に従ってサービス時間を決定しているため極端にサービス時間が短い場合があり，モニタに情報が届くことがあるからだと考えた．

5.6　分布ごとの最適な q_0 (AoI)

q_0 は各シンボルのサービス時間を決定する値である．すなわち，符号器の設定と言える．この設定で AoI が変化するため最適な値を求めたくなった．そこで式 (22) の q_0 の最適な値を求めようとしたが q_0 の式で解くことが難しかったため，ニュートン法を用い数値計算をした．これにより得られた各確率ごとの最適な q_0 を表したグラフを図 14 に示す．λ=0.5 のときはずっと下に凸のグラフなので最適な q_0 は滑らかに変わっていることが分かる．λ=1.0 のときは 5.2 節より p_0=0.5 のとき最小値をとる点が 2 つあることが分かっている．このグラフでは

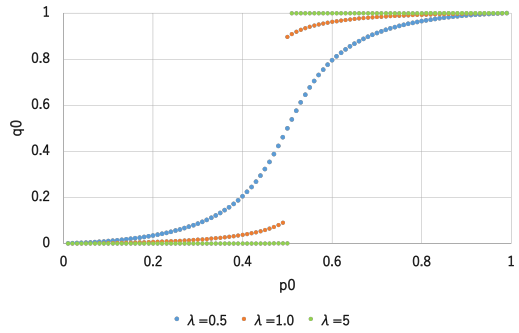

図 14　分布ごとの最適な q_0(AoI)

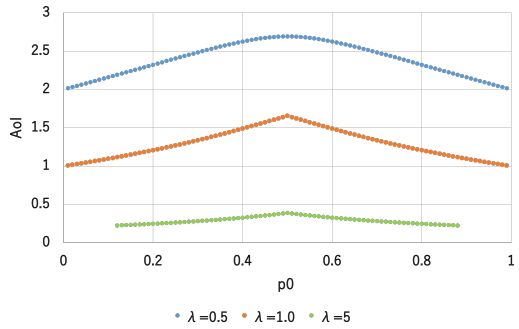

図 15　分布ごとの最小 AoI

p_0=0.5 のとき最適な q_0 の点が 2 つある．よって，p_0=0.5 を境にジャンプするようなグラフになっている．λ=5 のときはほとんど最適な q_0 の値が 0 と 1 であったが，これも p_0=0.5 を境に値がジャンプしている。これらの結果より λ が大きくなると最適な q_0 は 0 か 1 に大きく近づくことが分かった．最適な q_0 のときは最小の AoI になる．分布ごとの最小 AoI を図 15 示す．どのグラフも p_0=0.5 のとき一番大きい値をとっており，左右対称なグラフとなっている．これはクラフトの不等式の一方の符号語長が短いともう一方が長くなるような性質があるからだと考えられる．このような方法で分布ごとの最小 AoI を求められた．したがって，AoI を数理的に求められたと言える．

5.7　分布ごとの最適な q_0 (棄却率)

前節と同様に棄却率でも最適な q_0 の値に興味がある．しかし，式 (23) も解くことが難しかったためニュートン法を用い数値計算をした．得たグラフを図 16 に示す．式 (23) は λ=1.0 のときのみ扱えるのでグラフも 1 つとなる．5.3 節より p_0=0.5 のとき最小値をとる点が 2 つ

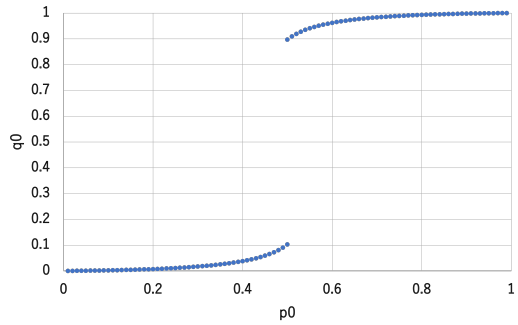

図 16　分布ごとの最適な q_0(棄却率)

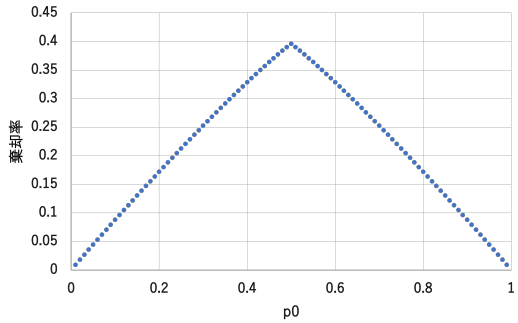

図 17　分布ごとの最小棄却率

あることが分かっている．前節と同様に p_0=0.5 のとき最適な q_0 は 2 つあり，ジャンプするようなグラフになっている．λ が同じだと AoI と棄却率の最適な符号は近いことが分かった．最適な q_0 のときは最小の棄却率になる．分布ごとの最小棄却率を図 17 に示す．グラフの特徴は前節の最小 AoI と同様の考察ができる．このような方法で分布ごとの最小棄却率を求められた．したがって，棄却率を数理的に求められたと言える．

6　まとめ

本研究では AoI という視点から見たサーバで割り込みが発生する符号器を追加した通信システムの数理的解析を行った．また，サーバで割り込みが発生するため割り込みの頻度である棄却率についても解析をした．時間平均 AoI と棄却率はシンボル ‘0’ の発生確率が等確率に近づくと増加し，分布に偏りがある場合，到着レート λ を大きくしたとき減少した．分布ごとの最小 AoI，最小棄却率を求めたことによりこの通信システムの数理的解析ができた．今後の課題として，バッファ付きのシステム，情報源のシンボルの数を増やすなどの拡張をして解析す

ることが挙げられる.

謝辞

本研究に際して，様々なご指導をしていただいた指導教員の西新幹彦先生，また助言をしていただいた研究室の先輩方に深く感謝申し上げます.

参考文献

[1] 井上文彰，滝根哲哉，「Age of Information (AoI) 基本概念と研究動向」，http://www.ieice.org/ess/sita/forum/article/2018/201810111910.pdf，2022 年 1 月閲覧.

[2] Y. Inoue, H. Masuyama, T. Takine and T. Tanaka, “A general formula for the stationary distribution of the age of information and its application to single-server queues”, IEEE Trans. Inf. Theory, vol.65, no.12, pp.8305–8324, Dec. 2019.

[3] 細海俊介,「情報鮮度の視点から見たバッファで割り込みのある待ち行列通信システムの実験的考察」，信州大学工学部電子情報システム工学科学士論文 (指導教員：西新幹彦) ，2022 年 2 月.

[4] S. Kaul, R. Yates and M. Gruteser, “Real-time status: How often should one update?”, in Proc. of IEEE INFOCOM, pp.2731–2735, Mar. 2012.

[5] R. D. Yates and S. K. Kaul, “The age of information: Real-time status updating by multiple sources”, IEEE Trans. Inf. Theory, vol.64, no.3, pp.1807–1827, Mar. 2019.

[6] 西新幹彦,「ポアソン過程に従ってシンボルを出力する情報源の符号化について」，The 26th Symposium on Information Theory and Its Applications(SITA2003) Higashiura, Hyogo, Japan, Dec. 15-18, 2003.

[7] 千葉直紀,「情報鮮度の観点に基づくバッファで割り込みがある通信システムの実験的評価」，信州大学大学院総合理工学研究科修士論文（指導教員：西新幹彦），2021 年 2 月.

[8] mt19937ar:Mersenne Twister with improved initialization, http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/MT2002/mt19937ar.html, 2022 年 1 月閲覧.

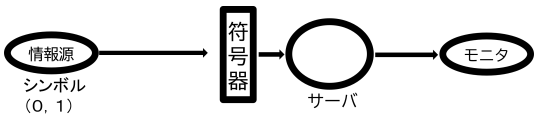

図 18　符号器を追加したサーバで割り込みが発生する通信システム

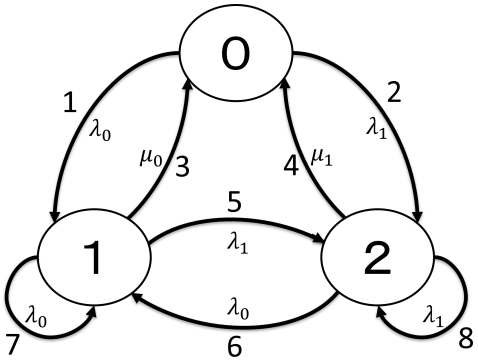

図 19　符号器を追加したサーバで割り込みが発生する通信システムの状態遷移図

付録 A　棄却率の導出

棄却率の導出方法は参考文献 [6] に習い求めた．この棄却率の求め方は情報源がポアソン過程に従うようなとき成り立つ．ここで符号器を追加したサーバで割り込みが発生する通信システムを例に挙げ棄却率を導出する．

A.1　符号器を追加したサーバ割り込みが発生する通信システムの棄却率導出

符号器を追加したサーバ割り込みが発生する通信システムを図 18 に示す．情報源について，シンボルがポアソン過程 $N(t)$ で到着すると考え，到着レート λ は 1.0 とし，シンボルの種類は簡単のため ‘0’，‘1’ の 2 種類とする．シンボルはバッファまでポアソン到着し，符号器で符号語に変換されサーバで処理される．サーバでのサービス時間は符号語長を期待値とした指数分布によって決定する．このようなシステムの状態遷移図を描くと図 19 のようになる．符号として語頭符号を使用する．したがって，符号語長はクラフトの不等式に従う．以下でクラフトの不等式及びその使用方法について説明する．

定理 1．クラフトの不等式

符号語長が $l_0, l_1 ... l_n$ である瞬時復号可能な符号が存在するための必要十分条件は，n を符号語の数，r を符号の種類数とすると，

$$\sum_{i=0}^{n} r^{-l_i} \leq 1 \tag{26}$$

を満たすことである．

この定理 1 を使い，計算の簡略化のために

$$\sum_{x \in \mathscr{X}} e^{-l(x)} = 1 \tag{27}$$

とした．$\mathscr{X}$ はシンボルの集合，$l(x)$ は定理 2 を満たす符号語長の関数である．シンボルは 2 種類なので，

$$e^{-l_0} + e^{-l_1} = 1 \tag{28}$$
$$q_0 + q_1 = 1 \tag{29}$$
$$l_0 = -\log q_0 \tag{30}$$
$$l_1 = -\log q_1 \tag{31}$$

とできる．

$S(l)$ を平均 l の指数分布に従う確率変数，通信路は単位時間当たり長さ R の符号語を送る能力があるとすると，パラメータ λ' に従う指数分布 $f(x)$ は

$$f(X) = \lambda' e^{-\lambda' x} \tag{32}$$

で表され，期待値 $E[X]$ は

$$E[X] = \frac{1}{\lambda'} = \frac{1}{R} l(X) \tag{33}$$

となる．よって棄却率 ε は

$$\varepsilon = \Pr\left\{N\Big(S\Big(\frac{1}{R}l(X)\Big)\Big) > 0\right\} \tag{34}$$

$$= 1 - \Pr\left\{N\Big(S\Big(\frac{1}{R}l(X)\Big)\Big) = 0\right\} \tag{35}$$

$$= 1 - \sum_{x\in\mathscr{X}} \Pr\left\{N\Big(S\Big(\frac{1}{R}l(X)\Big)\Big) = 0, X = x\right\} \tag{36}$$

$$= 1 - \sum_{x\in\mathscr{X}} \Pr\left\{N\Big(S\Big(\frac{1}{R}l(X)\Big)\Big) = 0\right\} P_X(x) \tag{37}$$

$$= 1 - \sum_{x\in\mathscr{X}} P_X(x) \int_0^\infty \Pr\left\{N\Big(S\Big(\frac{1}{R}l(X)\Big)\Big) = 0, S\Big(\frac{1}{R}l(x)\Big) = t\right\} dt \tag{38}$$

$$= 1 - \sum_{x\in\mathscr{X}} P_X(x) \int_0^\infty \Pr\left\{S\Big(\frac{1}{R}l(X)\Big) = t\right\} Pr\{N(t) = 0\} dt \tag{39}$$

$$= 1 - \sum_{x\in\mathscr{X}} P_X(x) \int_0^\infty \frac{R}{l_x} e^{-(\frac{R}{l_x}+\lambda)} dt \tag{40}$$

$$= 1 - \sum_{x\in\mathscr{X}} P_X(x) \frac{R}{l_x} \frac{l_x}{R + \lambda l_x} \tag{41}$$

$$= 1 - \sum_{x\in\mathscr{X}} P_X(x) \frac{1}{1 + \frac{\lambda}{R} l_x} \tag{42}$$

となる. シンボルは ‘0’, ‘1’ なので,

$$\varepsilon = 1 - \left(\frac{p_0}{1 + \frac{\lambda}{R} l_0} + \frac{p_1}{1 + \frac{\lambda}{R} l_1} \right) \tag{43}$$

となる．p_0，p_1 はシンボル ‘0’，‘1’ の発生確率である．このシステムでは $\frac{\lambda}{R}$ を 1 として考えられるので，式 (30) を代入し，

$$\varepsilon = 1 - \left(\frac{p_0}{1 - \log q_0} + \frac{p_1}{1 - \log (1 - q_0)} \right) \tag{44}$$

棄却率 ε を求められる．

付録 B　AoI の導出

AoI の導出方法は参考文献 [5] に習い求めた．手順を大きく分けると 3 つの工程に分けられる．まず，状態遷移図を描く．次に，各遷移のパラメータを求める．パラメータとはシンボルの到着レート，サーバでのサービスレート，定理で扱うベクトルのことである．最後に，定理

表 4　図 19 のマルコフ連鎖の遷移表

l	$q_l \to q_l'$	$\lambda^{(l)}$	$\boldsymbol{x}\boldsymbol{A}_l$	$\boldsymbol{A}_l$	$\boldsymbol{v}_{q_l}\boldsymbol{A}_l$
1	0→1	λ_0	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{00}\ 0]$
2	0→2	λ_1	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{00}\ 0]$
3	1→0	μ_0	$[x_1\ *]$	$\left[\begin{smallmatrix}0&*\\1&*\end{smallmatrix}\right]$	$[v_{11}\ *]$
4	2→0	μ_1	$[x_1\ *]$	$\left[\begin{smallmatrix}0&*\\1&*\end{smallmatrix}\right]$	$[v_{21}\ *]$
5	1→2	λ_1	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{10}\ 0]$
6	2→1	λ_0	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{20}\ 0]$
7	1→1	λ_0	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{10}\ 0]$
8	2→2	λ_1	$[x_0\ 0]$	$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right]$	$[v_{20}\ 0]$

に基づき AoI を導出する．ここで符号器を追加したサーバで割り込みが発生する通信システムを例に挙げ AoI を導出する．

B.1　符号器を追加したサーバ割り込みが発生する通信システムの AoI 導出

符号器を追加したサーバ割り込みが発生する通信システムを図 18 に示す．情報源について，シンボルがポアソン過程で到着すると考え，シンボルの種類は簡単のため ‘0’，‘1’ の 2 種類とする．シンボルはバッファまでポアソン到着し，符号器で符号語に変換されサーバで処理される．サーバでのサービス時間は符号語長を期待値とした指数分布によって決定する．このようなシステムの状態遷移図を描くと図 19 のようになる．状態 0 はサーバが空の状態，状態 1 はサーバでシンボル ‘0’ が処理されている状態，状態 2 はサーバでシンボル ‘1’ が処理されている状態を表している．ここで遷移番号 1 はサーバが空の状態からサーバにシンボル ‘0’ が到着したときの遷移を表しており，遷移番号 3 はサーバでシンボル ‘0’ のサービスが終了したことを表している．また，遷移番号 7，8 ではサーバで処理しているシンボルと同じシンボルが到着したことを表し，自己遷移が発生している. 状態遷移図を描くと，表 4 のような表を得られる．l は遷移の番号，q_l は遷移前の状態，q_l' は遷移後の状態，$\lambda^{(l)}$ は遷移のレート，$\boldsymbol{x}$ は x_0，x_1 からなるベクトルであり，x_0 はモニタにあるシンボルの年齢，x_1 はサーバにあるシンボルの年齢を表している．$\boldsymbol{A}_l$ は写像をする際に扱うベクトル，$\boldsymbol{v}_{q_l}$ は $v_{q_l 0}$，$v_{q_l 1}$ からなる定理で用いるベクトルである．$*$ は AoI を求めるときに無関係であることを表している．そして，AoI を導出する定理は以下である．

定理 2．AoI を導出する定理
Q を状態の集合，$\bar{q}$ を遷移後の状態，$\boldsymbol{b}_{\bar{q}}$ を各状態の傾き，ここでの傾きとはシンボルがサーバ

またはモニタなどで年齢をとることを指すので傾きは 0 か 1 である．$\boldsymbol{b}_0 = [1\ 0]$ は状態 0 のときモニタの傾きが 1 でサーバの傾きが 0 ということを表している．そして，AoI を Δ とすると，

$$\bar{\boldsymbol{v}}_{\bar{q}} \sum_{l \in \mathcal{L}_{\bar{q}}} \lambda^{(l)} = \boldsymbol{b}_{\bar{q}} \bar{\pi}_{\bar{q}} + \sum_{l \in \mathcal{L}'_{\bar{q}}} \lambda^{(l)} \bar{\boldsymbol{v}}_{ql} \boldsymbol{A}_l \quad \bar{q} \in Q \tag{45}$$

$$\Delta = \sum_{\bar{q} \in Q} \bar{v}_{\bar{q}0} \tag{46}$$

となる．$\bar{\pi}_{\bar{q}}$ は定常状態のときに成り立つ式であり以下のようになる．

$$\bar{\pi}_{\bar{q}} \sum_{l \in \mathcal{L}_{\bar{q}}} \lambda^{(l)} = \sum_{l \in \mathcal{L}'_{\bar{q}}} \lambda^{(l)} \bar{\pi}_{ql} \quad \bar{q} \in Q \tag{47}$$

$$\sum_{\bar{q} \in Q} \bar{\pi}_{\bar{q}} = 1 \tag{48}$$

この定理 2 を用いると

$$[v_{00}\ v_{01}](\lambda_0 + \lambda_1) = [1\ 0]\bar{\pi}_0 + \mu_0[v_{11}\ *] + \mu_1[v_{21}\ *] \tag{49}$$

$$[v_{10}\ v_{11}](\mu_0 + \lambda_0 + \lambda_1) = [1\ 1]\bar{\pi}_1 + \lambda_0[v_{00}\ 0] + \lambda_0[v_{10}\ 0] + \lambda_0[v_{20}\ 0] \tag{50}$$

$$[v_{20}\ v_{21}](\mu_1 + \lambda_0 + \lambda_1) = [1\ 1]\bar{\pi}_2 + \lambda_1[v_{00}\ 0] + \lambda_1[v_{10}\ 0] + \lambda_1[v_{20}\ 0] \tag{51}$$

$$\bar{\pi}_0(\lambda_0 + \lambda_1) = \mu_0\bar{\pi}_1 + \bar{\pi}_2 \tag{52}$$

$$\bar{\pi}_1(\mu_0 + \lambda_0 + \lambda_1) = \lambda_0\bar{\pi}_0 + \lambda_0\bar{\pi}_1 + \lambda_0\bar{\pi}_2 \tag{53}$$

$$\bar{\pi}_2(\mu_1 + \lambda_0 + \lambda_1) = \lambda_1\bar{\pi}_0 + \lambda_1\bar{\pi}_1 + \lambda_1\bar{\pi}_2 \tag{54}$$

このように式を立てられる．この式を解くと，

$$\Delta = \frac{1}{\lambda_0\mu_0 + \lambda_1\mu_1 + \mu_0\mu_1}\left(\lambda + \mu_0 + \mu_1 + \frac{\mu_0\mu_1}{\lambda}\right) \tag{55}$$

このような AoIΔ を求められる．このときの λ，λ_0，λ_1 は

$$\lambda = \lambda_0 + \lambda_1 \tag{56}$$

$$\lambda_0 = p_0\lambda \tag{57}$$

$$\lambda_1 = p_1\lambda \tag{58}$$

である．

付録 C　ソースコード

C.1　サーバでのサービス時間が指数分布に従う通信モデルをあらわしたプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "MT.h"

/*サーバで割り込みが起こるシステム log の底は e*/



double OCCUR_PROBABILITY_0 = 0.0;       //0 が出る確率

/*lambda に従う指数分布*/
double Random_Exp(double Lambda) {
    return -log(genrand_real2()) / Lambda;
}

/*確率 p で 0 を乱数生成*/
int Random_Binary(double Probability) {
    int Symbol;
    if ((double)genrand_real2() >= (1.0 - Probability)) {
        Symbol = 0;
    }
    else {
        Symbol = 1;
    }
    return Symbol;
}

/*100 万回まわして平均 AoI を出す関数（サービスレートは指数分布に従い決定）*/
double AoI_Func(double Kraft_0) {

    /*クラフトの不等式によりパラメータ導出*/
    double Kraft_Exp_0;
    double Kraft_Exp_1;
    double Kraft_1;
    double Service_Rate_0;
    double Service_Rate_1;
    Kraft_1 = 1 - Kraft_0;
    Kraft_Exp_0 = log(Kraft_0);
    Kraft_Exp_1 = log(Kraft_1);
    Service_Rate_0 = -1 / Kraft_Exp_0;
    Service_Rate_1 = -1 / Kraft_Exp_1;

    /*初期設定*/
    double Time = 0.0;                  //現在時刻
    double Next_Arrive_Time = 0.0;      //次の到着
    double Service_End_Time = -1.0;     //サービス終了時刻（ないとき-1.0）
    double Occur_Srv_Time = 0.0;        //サーバ処理中の情報の発生時刻
    //double Occur_Buf_Time = 0.0;        //バッファの中の情報の発生時刻
    int Count_Arrival = 0;              //到着数
    double Count_Occur = 0.0;           //発生数
#define NUM 1000000                         //上限値
    int Type_Srv = -1;                  //サーバの中身（0 or 1 or 空なら-1）
    //int Type_Buf = -1;                  //バッファの中身（0 or 1 or 空なら-1）



    /*変更するレート*/
```

```
#define ARRIVE_RATE 1000                //到着レート

    //double OCCUR_PROBABILITY_1;       //１が出る確率

/*AoI 計算必要*/
    double Occur_Time = 0.0;                //発生時刻
    double Display_Time = 0.0;              //表示時刻
    double Occur_One_Before_Time = 0.0;     //ひとつ前の発生時刻
    double Display_One_Before_Time = 0.0;   //ひとつ前の表示時刻
    double AoI = 0.0;                       //AoI
    double Total_AoI = 0.0;                 //AoI 全体を足したもの
    double Ave_AoI = 0.0;                   // 平均 AoI
    double First_Display_Time = 0.0;        //最初の表示時刻
    double Last_Display_Time = 0.0;         //最後の表示時刻


    init_genrand(19991202);                                                 //乱数の初期化
    Next_Arrive_Time = Random_Exp(ARRIVE_RATE);                             //最初の到着の決定
    while (Next_Arrive_Time >= 0 || Service_End_Time >= 0) {                //次の到着またはサービス終了時刻が 0 以
上の時 while 文をまわす
        if (Count_Arrival == NUM) {                                         //Count_Arrival = NUM になったら
ループから抜け出す
            Ave_AoI = Total_AoI / (Display_Time - First_Display_Time); //平均 AoI の導出
            //printf("break\n");
            break;
        }
        else if (Type_Srv == -1 || Service_End_Time > Next_Arrive_Time) {  //この条件通りなら到着処理
            Time = Next_Arrive_Time;                                        //現在時刻更新
            Occur_Srv_Time = Time;                                  //サーバー内の発生時刻決定
            Type_Srv = Random_Binary(OCCUR_PROBABILITY_0);          //乱数振って情報決定
            Count_Occur++;
            if (Type_Srv == 0) {
                Service_End_Time = Time + Random_Exp(Service_Rate_0);  //サービス終了時刻決定　０の時
                //printf("サービス終了時刻決定０\n");
        }
            else if (Type_Srv == 1) {
                Service_End_Time = Time + Random_Exp(Service_Rate_1);  //サービス終了時刻決定　１の時
                //printf("サービス終了時刻決定 1\n");
            }
            Next_Arrive_Time = Time + Random_Exp(ARRIVE_RATE);              //次の到着の決定



        }


        else {                                                              //サービス終了処理
            Time = Service_End_Time;                                        //現在時刻更新
            Count_Arrival++;                                                //到着数＋＋
            Type_Srv = -1;
            //printf("サービス終了 shori\n");


            /*AoI 導出関係*/
            Occur_One_Before_Time = Occur_Time;
            Occur_Time = Occur_Srv_Time;
            Display_One_Before_Time = Display_Time;
            Display_Time = Time;
            if (Occur_One_Before_Time == 0.0) {
                First_Display_Time = Display_Time;
            }
            if (Occur_One_Before_Time != 0.0) {
                AoI = 0.5 * (pow(Display_Time - Occur_One_Before_Time, 2) -
                (pow(Display_One_Before_Time - Occur_One_Before_Time, 2)));
                Total_AoI += AoI;
            }


        }
```

```
    }
    //printf("発生数 %f\n", Count_Occur);
    //printf("到着数 %d\n", Count_Arrival);
    //printf("現在時刻 %f\n", Time);
    //printf("サーバ中身 %d\n", Type_Srv);
    //printf("次の到着時間 %f\n", Next_Arrive_Time);
    //printf("サービス終了時刻 %f\n", Service_End_Time);
    printf("%f\n", Display_Time);
    return Ave_AoI;


}

/*100 万回まわして棄却率を出す関数 (サービスレートは指数分布に従い決定)*/
double Rejection_Rate_Func(double Kraft_0) {

    /*クラフトの不等式によりパラメータ導出*/
    double Kraft_Exp_0;
    double Kraft_Exp_1;
    double Kraft_1;
    double Service_Rate_0;
    double Service_Rate_1;
    Kraft_1 = 1 - Kraft_0;
    Kraft_Exp_0 = log(Kraft_0);
    Kraft_Exp_1 = log(Kraft_1);
    Service_Rate_0 = -1 / Kraft_Exp_0;
    Service_Rate_1 = -1 / Kraft_Exp_1;

    /*初期設定*/
    double Time = 0.0;                //現在時刻
    double Next_Arrive_Time = 0.0;    //次の到着
    double Service_End_Time = -1.0;   //サービス終了時刻（ないとき-1.0）
    double Occur_Srv_Time = 0.0;      //サーバ処理中の情報の発生時刻
    //double Occur_Buf_Time = 0.0;      //バッファの中の情報の発生時刻
    int Count_Arrival = 0;            //到着数
    double Count_Occur = 0.0;         //発生数
#define NUM 1000000                       //上限値
    int Type_Srv = -1;                //サーバの中身（0 or 1 or 空なら-1）
    //int Type_Buf = -1;                //バッファの中身（0 or 1 or 空なら-1）




    /*変更するレート*/
#define ARRIVE_RATE 1.0               //到着レート

    //double OCCUR_PROBABILITY_1;       //１が出る確率

/*AoI 計算必要*/
    double Occur_Time = 0.0;              //発生時刻
    double Display_Time = 0.0;            //表示時刻
    double Occur_One_Before_Time = 0.0;   //ひとつ前の発生時刻
    double Display_One_Before_Time = 0.0; //ひとつ前の表示時刻
    double AoI = 0.0;                     //AoI
    double Total_AoI = 0.0;               //AoI 全体を足したもの
    double Ave_AoI = 0.0;                 // 平均 AoI
    double First_Display_Time = 0.0;      //最初の表示時刻
    double Last_Display_Time = 0.0;       //最後の表示時刻
    double Rejection_Rate = 0.0;          //棄却率


    init_genrand(19991202);                                                 //乱数の初期化
    Next_Arrive_Time = Random_Exp(ARRIVE_RATE);                             //最初の到着の決定
    while (Next_Arrive_Time >= 0 || Service_End_Time >= 0) {                //次の到着またはサービス終了時刻が 0 以
上の時 while 文をまわす
        if (Count_Arrival == NUM) {                                         //Count_Arrival = NUM になったら
ループから抜け出す
            Ave_AoI = Total_AoI / (Display_Time - First_Display_Time); //平均 AoI の導出
```

```
            Rejection_Rate = (Count_Occur - NUM) / Count_Occur;         //棄却率の導出
            break;
        }
        else if (Type_Srv == -1 || Service_End_Time > Next_Arrive_Time) {  //この条件通りなら到着処理
        Time = Next_Arrive_Time;                                          //現在時刻更新
            Occur_Srv_Time = Time;                                      //サーバー内の発生時刻決定
            Type_Srv = Random_Binary(OCCUR_PROBABILITY_0);              //乱数振って情報決定
            Count_Occur++;
            if (Type_Srv == 0) {
                Service_End_Time = Time + Random_Exp(Service_Rate_0);  //サービス終了時刻決定　0の時
                //printf("サービス終了時刻決定0\n");
            }
            else if (Type_Srv == 1) {
            Service_End_Time = Time + Random_Exp(Service_Rate_1);  //サービス終了時刻決定　1の時
                //printf("サービス終了時刻決定 1\n");
            }
            Next_Arrive_Time = Time + Random_Exp(ARRIVE_RATE);              //次の到着の決定



        }


        else {                                                              //サービス終了処理
            Time = Service_End_Time;                                        //現在時刻更新
            Count_Arrival++;                                                //到着数＋＋
            Type_Srv = -1;
            //printf("サービス終了 shori\n");


            /*AoI 導出関係*/
            Occur_One_Before_Time = Occur_Time;
            Occur_Time = Occur_Srv_Time;
            Display_One_Before_Time = Display_Time;
            Display_Time = Time;
            if (Occur_One_Before_Time == 0.0) {
                First_Display_Time = Display_Time;
            }
            if (Occur_One_Before_Time != 0.0) {
                AoI = 0.5 * (pow(Display_Time - Occur_One_Before_Time, 2) -
                (pow(Display_One_Before_Time - Occur_One_Before_Time, 2)));
                Total_AoI += AoI;
            }


        }
    }
    //printf("発生数 %f\n", Count_Occur);
    //printf("到着数 %d\n", Count_Arrival);
    //printf("現在時刻 %f\n", Time);
    //printf("サーバ中身 %d\n", Type_Srv);
    //printf("次の到着時間 %f\n", Next_Arrive_Time);
    //printf("サービス終了時刻 %f\n", Service_End_Time);
    return Rejection_Rate;


}

/*最小の AoI を求める関数*/
double Min_AoI_Func(double  OCCUR_PROBABILITY_0) {

    double AoI[5] = { AoI_Func(0.01),AoI_Func(0.255),AoI_Func(0.5),AoI_Func(0.745),AoI_Func(0.99) };     //
五点探索（仮）の配列の初期化
    double Kraft_0_AoI_Search_0 = 0.01;
    double Kraft_0_AoI_Search_1 = 0.255;
    double Kraft_0_AoI_Search_2 = 0.5;
    double Kraft_0_AoI_Search_3 = 0.745;
    double Kraft_0_AoI_Search_4 = 0.99;
```

```
int Count_AoI_Search = 0;

while (Count_AoI_Search < 20) {

    int mini_AoI = 1;
    if (AoI[2] < AoI[mini_AoI]) {
        mini_AoI = 2;
    }
    if (AoI[3] < AoI[mini_AoI]) {
        mini_AoI = 3;
    }

    switch (mini_AoI) {
    case 1:
        AoI[4] = AoI[2];
        AoI[2] = AoI[1];
        Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_2;
        Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_1;
        break;
    case 2:
        AoI[0] = AoI[1];
        AoI[4] = AoI[3];
        Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_1;
        Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_3;
        break;
    case 3:
        AoI[0] = AoI[2];
        AoI[2] = AoI[3];
        Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_2;
        Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_3;
        break;
    }
    Kraft_0_AoI_Search_1 = (Kraft_0_AoI_Search_0 + Kraft_0_AoI_Search_2) * 0.5;
    Kraft_0_AoI_Search_3 = (Kraft_0_AoI_Search_2 + Kraft_0_AoI_Search_4) * 0.5;
    AoI[1] = AoI_Func(Kraft_0_AoI_Search_1);
    AoI[3] = AoI_Func(Kraft_0_AoI_Search_3);
    Count_AoI_Search += 1;

    /*printf("%d 回目\n", Count_AoI_Search);
    printf("%d 番\n", mini_AoI);
    printf("------AoI------\n");
    printf("%f\n", AoI[0]);
    printf("%f\n", AoI[1]);
    printf("%f\n", AoI[2]);
    printf("%f\n", AoI[3]);
    printf("%f\n", AoI[4]);
    printf("------Kraft_0------\n");
    printf("%f\n", Kraft_0_AoI_Search_0);
    printf("%f\n", Kraft_0_AoI_Search_1);
    printf("%f\n", Kraft_0_AoI_Search_2);
    printf("%f\n", Kraft_0_AoI_Search_3);
    printf("%f\n", Kraft_0_AoI_Search_4);
    printf("-.-.-.-.-.-.--.--.-.--.-\n");*/



}

int mini_AoI = 1;
if (AoI[2] < AoI[mini_AoI]) {
    mini_AoI = 2;
}
if (AoI[3] < AoI[mini_AoI]) {
    mini_AoI = 3;
}

double Min_AoI = 0.0;
Min_AoI = AoI[mini_AoI];
return Min_AoI;
```

```
}

/*最小の棄却率を求める関数*/
double Min_Rejection_Rate_Func(double  OCCUR_PROBABILITY_0) {

    double Rejection_Rate[5] = { Rejection_Rate_Func(0.01),Rejection_Rate_Func(0.255),
Rejection_Rate_Func(0.5),Rejection_Rate_Func(0.745),Rejection_Rate_Func(0.99) };     //五点探索（仮）の配列
の初期化
    double Kraft_0_Rejection_Rate_Search_0 = 0.01;
    double Kraft_0_Rejection_Rate_Search_1 = 0.255;
    double Kraft_0_Rejection_Rate_Search_2 = 0.5;
    double Kraft_0_Rejection_Rate_Search_3 = 0.745;
    double Kraft_0_Rejection_Rate_Search_4 = 0.99;

    int Count_Rejection_Rate_Search = 0;

    while (Count_Rejection_Rate_Search < 20) {

        int mini_Rejection_Rate = 1;
        if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 2;
        }
        if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 3;
        }

        switch (mini_Rejection_Rate) {
        case 1:
            Rejection_Rate[4] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[1];
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_2;
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_1;
            break;
        case 2:
            Rejection_Rate[0] = Rejection_Rate[1];
            Rejection_Rate[4] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_1;
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_3;
            break;
        case 3:
            Rejection_Rate[0] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_2;
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_3;
            break;
        }
        Kraft_0_Rejection_Rate_Search_1 = (Kraft_0_Rejection_Rate_Search_0 + Kraft_0_Rejection_Rate_Search_2) * 0.5;
        Kraft_0_Rejection_Rate_Search_3 = (Kraft_0_Rejection_Rate_Search_2 + Kraft_0_Rejection_Rate_Search_4) * 0.5;
        Rejection_Rate[1] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_1);
        Rejection_Rate[3] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_3);
        Count_Rejection_Rate_Search += 1;

        printf("%d 回目\n", Count_Rejection_Rate_Search);
        printf("%d 番\n", mini_Rejection_Rate);
        printf("------Rejection_Rate------\n");
        printf("%f\n", Rejection_Rate[0]);
        printf("%f\n", Rejection_Rate[1]);
        printf("%f\n", Rejection_Rate[2]);
        printf("%f\n", Rejection_Rate[3]);
        printf("%f\n", Rejection_Rate[4]);
        printf("------Kraft_0------\n");
        printf("%f\n", Kraft_0_Rejection_Rate_Search_0);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_1);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_2);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_3);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_4);
        printf("-.-.-.-.-.--.--.-.--.-\n");
```

```
    }

    int mini_Rejection_Rate = 1;
    if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 2;
    }
    if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 3;
    }
    //printf("棄却率最小値は  %f\n", Rejection_Rate[mini_Rejection_Rate]);
    double Min_Rejection_Rate = 0.0;
    Min_Rejection_Rate = Rejection_Rate[mini_Rejection_Rate];
    return Min_Rejection_Rate;
}

/*最小の AoI の時の Kraft_0 を求める関数*/
double Kraft_0_Min_AoI_Func(double  OCCUR_PROBABILITY_0) {
    double AoI[5] = { AoI_Func(0.01),AoI_Func(0.255),AoI_Func(0.5),
AoI_Func(0.745),AoI_Func(0.99) };     //五点探索（仮）の配列の初期化
    double Kraft_0_AoI_Search_0 = 0.01;
    double Kraft_0_AoI_Search_1 = 0.255;
    double Kraft_0_AoI_Search_2 = 0.5;
    double Kraft_0_AoI_Search_3 = 0.745;
    double Kraft_0_AoI_Search_4 = 0.99;

    int Count_AoI_Search = 0;

    while (Count_AoI_Search < 20) {

        int mini_AoI = 1;
        if (AoI[2] < AoI[mini_AoI]) {
            mini_AoI = 2;
        }
        if (AoI[3] < AoI[mini_AoI]) {
            mini_AoI = 3;
        }

        switch (mini_AoI) {
        case 1:
            AoI[4] = AoI[2];
            AoI[2] = AoI[1];
            Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_2;
            Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_1;
            break;
        case 2:
            AoI[0] = AoI[1];
            AoI[4] = AoI[3];
            Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_1;
            Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_3;
            break;
        case 3:
            AoI[0] = AoI[2];
            AoI[2] = AoI[3];
            Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_2;
            Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_3;
            break;
        }
        Kraft_0_AoI_Search_1 = (Kraft_0_AoI_Search_0 + Kraft_0_AoI_Search_2) * 0.5;
        Kraft_0_AoI_Search_3 = (Kraft_0_AoI_Search_2 + Kraft_0_AoI_Search_4) * 0.5;
        AoI[1] = AoI_Func(Kraft_0_AoI_Search_1);
        AoI[3] = AoI_Func(Kraft_0_AoI_Search_3);
        Count_AoI_Search += 1;

        /*printf("%d 回目\n", Count_AoI_Search);
        printf("%d 番\n", mini_AoI);
        printf("------AoI------\n");
        printf("%f\n", AoI[0]);
        printf("%f\n", AoI[1]);
```

```
        printf("%f\n", AoI[2]);
        printf("%f\n", AoI[3]);
        printf("%f\n", AoI[4]);
        printf("------Kraft_0------\n");
        printf("%f\n", Kraft_0_AoI_Search_0);
        printf("%f\n", Kraft_0_AoI_Search_1);
        printf("%f\n", Kraft_0_AoI_Search_2);
        printf("%f\n", Kraft_0_AoI_Search_3);
        printf("%f\n", Kraft_0_AoI_Search_4);
        printf("-.-.-.-.-.--.--.-.--.-\n");*/



    }
    double Kraft_0_Min_AoI = Kraft_0_AoI_Search_1;
    int mini_AoI = 1;
    if (AoI[2] < AoI[mini_AoI]) {
        mini_AoI = 2;
        Kraft_0_Min_AoI = Kraft_0_AoI_Search_2;
    }
    if (AoI[3] < AoI[mini_AoI]) {
        mini_AoI = 3;
        Kraft_0_Min_AoI = Kraft_0_AoI_Search_3;
    }

    return Kraft_0_Min_AoI;
}

/*最小の棄却率の時の Kraft_0 を求める関数*/
double Kraft_0_Min_Rejection_Rate_Func(double  OCCUR_PROBABILITY_0) {

    double Rejection_Rate[5] = { Rejection_Rate_Func(0.01),Rejection_Rate_Func(0.255),
Rejection_Rate_Func(0.5),Rejection_Rate_Func(0.745),Rejection_Rate_Func(0.99) };     //五点探索（仮）の配列
の初期化
    double Kraft_0_Rejection_Rate_Search_0 = 0.01;
    double Kraft_0_Rejection_Rate_Search_1 = 0.255;
    double Kraft_0_Rejection_Rate_Search_2 = 0.5;
    double Kraft_0_Rejection_Rate_Search_3 = 0.745;
    double Kraft_0_Rejection_Rate_Search_4 = 0.99;

    int Count_Rejection_Rate_Search = 0;

    while (Count_Rejection_Rate_Search < 20) {

        int mini_Rejection_Rate = 1;
        if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 2;
        }
        if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 3;
        }

        switch (mini_Rejection_Rate) {
        case 1:
            Rejection_Rate[4] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[1];
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_2;
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_1;
            break;
        case 2:
            Rejection_Rate[0] = Rejection_Rate[1];
            Rejection_Rate[4] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_1;
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_3;
            break;
        case 3:
            Rejection_Rate[0] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_2;
```

```
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_3;
            break;
        }
        Kraft_0_Rejection_Rate_Search_1 = (Kraft_0_Rejection_Rate_Search_0 + Kraft_0_Rejection_Rate_Search_2) * 0.5;
        Kraft_0_Rejection_Rate_Search_3 = (Kraft_0_Rejection_Rate_Search_2 + Kraft_0_Rejection_Rate_Search_4) * 0.5;
        Rejection_Rate[1] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_1);
        Rejection_Rate[3] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_3);
        Count_Rejection_Rate_Search += 1;

        /*printf("%d 回目\n", Count_Rejection_Rate_Search);
        printf("%d 番\n", mini_Rejection_Rate);
        printf("------Rejection_Rate------\n");
        printf("%f\n", Rejection_Rate[0]);
        printf("%f\n", Rejection_Rate[1]);
        printf("%f\n", Rejection_Rate[2]);
        printf("%f\n", Rejection_Rate[3]);
        printf("%f\n", Rejection_Rate[4]);
        printf("------Kraft_0------\n");
        printf("%f\n", Kraft_0_Rejection_Rate_Search_0);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_1);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_2);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_3);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_4);
        printf("-.-.-.-.-.--.--.-.--.-\n");*/



    }
    double Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_1;
    int mini_Rejection_Rate = 1;
    if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 2;
        Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_2;
    }
    if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 3;
        Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_3;
    }
    return Kraft_0_Min_Rejection_Rate;
}

/*最小の AoI の時の棄却率を求める関数*/
double Rejection_Rate_of_Min_AoI_Func(double  OCCUR_PROBABILITY_0) {

    double AoI[5] = { AoI_Func(0.01),AoI_Func(0.255),AoI_Func(0.5),AoI_Func(0.745),AoI_Func(0.99) };     //
五点探索（仮）の配列の初期化
    double Kraft_0_AoI_Search_0 = 0.01;
    double Kraft_0_AoI_Search_1 = 0.255;
    double Kraft_0_AoI_Search_2 = 0.5;
    double Kraft_0_AoI_Search_3 = 0.745;
    double Kraft_0_AoI_Search_4 = 0.99;

    int Count_AoI_Search = 0;

    while (Count_AoI_Search < 20) {

        int mini_AoI = 1;
        if (AoI[2] < AoI[mini_AoI]) {
            mini_AoI = 2;
        }
        if (AoI[3] < AoI[mini_AoI]) {
            mini_AoI = 3;
        }

        switch (mini_AoI) {
        case 1:
            AoI[4] = AoI[2];
            AoI[2] = AoI[1];
            Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_2;
```

```
            Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_1;
            break;
        case 2:
            AoI[0] = AoI[1];
            AoI[4] = AoI[3];
            Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_1;
            Kraft_0_AoI_Search_4 = Kraft_0_AoI_Search_3;
            break;
        case 3:
            AoI[0] = AoI[2];
            AoI[2] = AoI[3];
            Kraft_0_AoI_Search_0 = Kraft_0_AoI_Search_2;
            Kraft_0_AoI_Search_2 = Kraft_0_AoI_Search_3;
            break;
        }
        Kraft_0_AoI_Search_1 = (Kraft_0_AoI_Search_0 + Kraft_0_AoI_Search_2) * 0.5;
        Kraft_0_AoI_Search_3 = (Kraft_0_AoI_Search_2 + Kraft_0_AoI_Search_4) * 0.5;
        AoI[1] = AoI_Func(Kraft_0_AoI_Search_1);
        AoI[3] = AoI_Func(Kraft_0_AoI_Search_3);
        Count_AoI_Search += 1;

        /*printf("%d 回目\n", Count_AoI_Search);
        printf("%d 番\n", mini_AoI);
        printf("------AoI------\n");
        printf("%f\n", AoI[0]);
        printf("%f\n", AoI[1]);
        printf("%f\n", AoI[2]);
        printf("%f\n", AoI[3]);
        printf("%f\n", AoI[4]);
        printf("------Kraft_0------\n");
        printf("%f\n", Kraft_0_AoI_Search_0);
        printf("%f\n", Kraft_0_AoI_Search_1);
        printf("%f\n", Kraft_0_AoI_Search_2);
        printf("%f\n", Kraft_0_AoI_Search_3);
        printf("%f\n", Kraft_0_AoI_Search_4);
        printf("-.-.-.-.-.-.--.--.-.--.-\n");*/



    }
    double Kraft_0_Min_AoI = Kraft_0_AoI_Search_1;
    int mini_AoI = 1;
    if (AoI[2] < AoI[mini_AoI]) {
        mini_AoI = 2;
        Kraft_0_Min_AoI = Kraft_0_AoI_Search_2;
    }
    if (AoI[3] < AoI[mini_AoI]) {
        mini_AoI = 3;
        Kraft_0_Min_AoI = Kraft_0_AoI_Search_3;
    }

    return Rejection_Rate_Func(Kraft_0_Min_AoI);
}

/*最小の棄却率の時の AoI を求める関数*/
double AoI_of_Min_Rejection_Rate_Func(double  OCCUR_PROBABILITY_0) {

    double Rejection_Rate[5] = { Rejection_Rate_Func(0.01),Rejection_Rate_Func(0.255),
Rejection_Rate_Func(0.5),Rejection_Rate_Func(0.745),Rejection_Rate_Func(0.99) };     //五点探索（仮）の配列
の初期化
    double Kraft_0_Rejection_Rate_Search_0 = 0.01;
    double Kraft_0_Rejection_Rate_Search_1 = 0.255;
    double Kraft_0_Rejection_Rate_Search_2 = 0.5;
    double Kraft_0_Rejection_Rate_Search_3 = 0.745;
    double Kraft_0_Rejection_Rate_Search_4 = 0.99;

    int Count_Rejection_Rate_Search = 0;

    while (Count_Rejection_Rate_Search < 20) {
```

```
        int mini_Rejection_Rate = 1;
        if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 2;
        }
        if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
            mini_Rejection_Rate = 3;
        }

        switch (mini_Rejection_Rate) {
       case 1:
            Rejection_Rate[4] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[1];
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_2;
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_1;
            break;
       case 2:
            Rejection_Rate[0] = Rejection_Rate[1];
            Rejection_Rate[4] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_1;
            Kraft_0_Rejection_Rate_Search_4 = Kraft_0_Rejection_Rate_Search_3;
            break;
       case 3:
            Rejection_Rate[0] = Rejection_Rate[2];
            Rejection_Rate[2] = Rejection_Rate[3];
            Kraft_0_Rejection_Rate_Search_0 = Kraft_0_Rejection_Rate_Search_2;
            Kraft_0_Rejection_Rate_Search_2 = Kraft_0_Rejection_Rate_Search_3;
            break;
        }
        Kraft_0_Rejection_Rate_Search_1 = (Kraft_0_Rejection_Rate_Search_0 + Kraft_0_Rejection_Rate_Search_2) * 0.5;
        Kraft_0_Rejection_Rate_Search_3 = (Kraft_0_Rejection_Rate_Search_2 + Kraft_0_Rejection_Rate_Search_4) * 0.5;
        Rejection_Rate[1] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_1);
        Rejection_Rate[3] = Rejection_Rate_Func(Kraft_0_Rejection_Rate_Search_3);
        Count_Rejection_Rate_Search += 1;

        /*printf("%d 回目\n", Count_Rejection_Rate_Search);
        printf("%d 番\n", mini_Rejection_Rate);
        printf("------Rejection_Rate------\n");
        printf("%f\n", Rejection_Rate[0]);
        printf("%f\n", Rejection_Rate[1]);
        printf("%f\n", Rejection_Rate[2]);
        printf("%f\n", Rejection_Rate[3]);
        printf("%f\n", Rejection_Rate[4]);
        printf("------Kraft_0------\n");
        printf("%f\n", Kraft_0_Rejection_Rate_Search_0);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_1);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_2);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_3);
        printf("%f\n", Kraft_0_Rejection_Rate_Search_4);
        printf("-.-.-.-.-.--.--.-.--.-\n");*/



    }
    double Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_1;
    int mini_Rejection_Rate = 1;
    if (Rejection_Rate[2] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 2;
        Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_2;
    }
    if (Rejection_Rate[3] < Rejection_Rate[mini_Rejection_Rate]) {
        mini_Rejection_Rate = 3;
        Kraft_0_Min_Rejection_Rate = Kraft_0_Rejection_Rate_Search_3;
    }
    return AoI_Func(Kraft_0_Min_Rejection_Rate);
}
```

```
/*本文*/
int main(void) {


    OCCUR_PROBABILITY_0 = 0.1;
    double aaa = 0.1;
    while (aaa <= 0.99) {
        //printf("%f\n", Rejection_Rate_Func(aaa));
        printf("%f\n", AoI_Func(aaa));
        //printf("%f\n", aaa);
        aaa += 0.1;
    }

    //OCCUR_PROBABILITY_0 = 0.1;
    /*while (OCCUR_PROBABILITY_0 <= 0.51) {
        //printf("%f\n", OCCUR_PROBABILITY_0);
        //printf("%f\n", Rejection_Rate_of_Min_AoI_Func(OCCUR_PROBABILITY_0));
        //printf("%f\n", AoI_of_Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));
        //printf("%f\n", Min_AoI_Func(OCCUR_PROBABILITY_0));
        //printf("%f\n", Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));
        //printf("%f\n", Kraft_0_Min_AoI_Func(OCCUR_PROBABILITY_0));
        //printf("%f\n", Kraft_0_Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));


        OCCUR_PROBABILITY_0 += 0.1;
    }*/
    /*printf("%f\n", Min_AoI_Func(OCCUR_PROBABILITY_0));
    printf("%f\n", Kraft_0_Min_AoI_Func(OCCUR_PROBABILITY_0));
    printf("%f\n", Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));
    printf("%f\n", Kraft_0_Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));
    printf("%f\n", Rejection_Rate_of_Min_AoI_Func(OCCUR_PROBABILITY_0));
    printf("%f\n", AoI_of_Min_Rejection_Rate_Func(OCCUR_PROBABILITY_0));*/


}
```