信州大学工学部

学士論文

パケット間隔で情報を送る通信路に対する最尤復号のための距離関数

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　11T2801F
氏名　飯島 一貴

2015 年 3 月 10 日

目次

1 はじめに 1

1.1 研究背景と目的 1

1.2 本論文の構成 1

2 通信システム 1

2.1 通信路モデル 1

2.2 パケット間隔で情報を送る符号 2

2.3 達成可能性と通信路容量 3

3 符号器と復号器の設計 4

3.1 ポアソン過程による符号語の生成 5

3.2 距離関数に基づく復号 5

3.3 最尤復号のための距離関数 9

4 復号器の評価 11

4.1 評価方法 11

4.2 結果と考察 11

5 まとめ 14

謝辞 15

参考文献 15

付録 A ソースコード 16

A.1 符号語をパケット数に応じた個数だけ生成し
復号誤り回数を計測するプログラム 16

A.2 符号化レート R を入力した際に各パケット数 n に対する
復号誤り確率 ϵ を返すプログラム 21

1　はじめに

1.1　研究背景と目的

情報伝達方法の一つとしてパケット通信がある．パケット通信とはコンピュータ通信において、データを小さなまとまりに分割して一つ一つ送受信する通信方式である．分割されたデータはパケットと呼ばれ，多くの場合はメッセージを符号化し，その符号語をパケットに収める．パケット通信を使うと，送受信間の通信に途中の回線が占有されることがなくなり，通信回線を効率よく利用することができる．また，柔軟に経路選択が行えるため，ネットワークの一部に障害が出たとしても他の回線で代替することができる，という利点もある．

通常のパケット通信ではパケットの中に伝えるべき情報が収められている．しかしながら，Anantharam and Verdú[1] によると，パケット間隔もまた情報を運ぶことができる．つまり，パケットの送信間隔の長短に意味を与えておくことで受信者はパケットの到着間隔から情報を得ることができる．パケット通信を行う際，ネットワーク内では様々な処理時間の変化や転送経路の変動等が発生する．これによりパケットの時間間隔が送信時と受信時で異なってしまい，復号器では誤り訂正が必要になる．

大きさのない空のパケットの送信間隔に情報を載せたときの，単一サーバ待ち行列システムの通信路容量は，文献 [1] で既に求められている．しかし，通信路容量を達成するための現実的な方法は未だ知られていない．そこで，それを達成できるような符号を構成することを本研究において大きな目標とする．本論文では，ランダム符号化で通信路容量を達成する最尤推定のための距離関数を定義し，復号誤り確率を実験的に検証する．

1.2　本論文の構成

本論文は次のような章から成る．2 章では本研究で扱う通信システムの説明や，符号の定義について述べる．3 章では符号語の生成方法と，従来研究を含む 3 種類の距離関数による復号方法を説明し，本研究で扱う通信路における最尤復号法についても説明する．4 章では 3 種類の距離関数による復号の評価を行い，考察する．5 章でまとめる．

2　通信システム

2.1　通信路モデル

インターネットのようなネットワークでは複数の送信者が不規則にパケットを送信する．パケットは複数のサーバに中継されて最終目的地に届けられる．パケットが送信者のもとを出発

してから目的地に到着するまでの時間は，他のユーザがどの程度パケットを送信しているか，すなわちネットワークのトラフィックの状態に依存する．いま，一対の送受信者に着目すると，ネットワーク全体はひとつの複雑な待ち行列システムとみなすことができる．本来ならパケットの中にも情報を載せるのだが，本研究では簡単化するためにパケットの中の情報量は 0 とし，パケットの送信間隔のみで通信することを考える．このような単一サーバによって構成された FIFO 待ち行列システムのモデルを図 1 に示す．

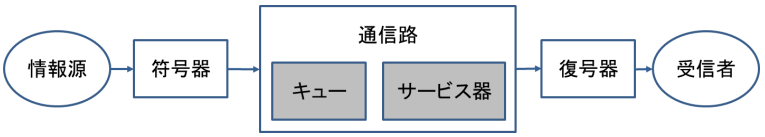

図 1　通信路モデル

符号器では送信対象のメッセージをパケットの時間間隔の列に変換し，通信路へ送信する．通信路は単一のキューとサービス器により構成される．サービス器は指数分布に従う時間でサービスを行うと仮定し，サービスレート μ[packet/sec] の性能を持っているとする．単一サーバ待ち行列システムを考えているため，一つのパケットがサービスを受けている途中で到着したパケットを順序を崩すことなく待たせるために，サービス器の手前にキューを設置する．このキューは待機するパケットをすべて格納できるように十分な大きさがあると仮定する．復号器では到着したパケットの時間間隔からメッセージを復号する．この符号器 · 復号器のペアを符号と呼ぶ．

この通信路はサービスレート μ のみで特徴づけられ，一般性を損なうことなく $\mu = 1$ の場合を考察すれば十分である．なぜなら，一般の場合は $\mu = 1$ のときに得られたパケットの送信間隔を $\frac{1}{\mu}$ 倍にしたものを符号語とすれば同じ誤り確率が得られるからである．

2.2　パケット間隔で情報を送る符号

形式的には，符号語は非負実数の並びである．本研究では従来研究 [1] に倣い，パケットの間隔を用いた符号を次のように定義する．

定義 1　非負実数ベクトル $(a_1, a_2, \cdots, a_n)$ を符号語という．符号器が符号語 $(a_1, a_2, \cdots, a_n)$ を送信するとき，時刻 0 で空のキューに対して時刻 $\sum_{i=1}^{k} a_i$ に k 番目 $(k \geq 1)$ の到着が起こる．符号器は M 個の符号語を持っているとし，それらは等確率で選ばれて送信されるとする．復号器は n 個の出発を観測し，正しい符号語を平均確率 $1 - \epsilon$ で復号するとする．最後の出発が起こる時刻の平均を T とおく．このような符号器 · 復号器を (n, M, T, ϵ) 符号という．この

符号の符号化レートを $(\log M)/T$ と定める[*1]．符号化レートとは単位時間あたりに符号器が送信する情報量を表している．

2.3 達成可能性と通信路容量

通信において符号化レートは大きいほうがよいが，復号器で正しく復号されなければ意味がない．したがって，与えられた符号化レートに対してほぼ確実に正しく復号できるような復号器が存在するかどうかが重要である．この概念を次のように定義する．以下に示す 2 つの定義は Anantharam and Verdú[1] にもとづいて文献 [2] で定義されたものである．

定義 2　符号化レート R が**達成可能**とは

$$\liminf_{n\to\infty} \frac{\log M_n}{T_n} \geq R \tag{1}$$

$$\lim_{n\to\infty} \frac{n}{T_n} > 0 \tag{2}$$

$$\lim_{n\to\infty} \epsilon_n = 0 \tag{3}$$

となるような符号の列 $\{(n, M_n, T_n, \epsilon_n)\}_{n=1}^{\infty}$ が存在することである．通信路容量を

$$C \triangleq \sup\{R|\ \text{符号化レート } R \text{ は達成可能.}\}$$

と定める．

定義 3　符号化レート R が**出力レート λ で ϵ-達成可能**とは

$$\liminf_{n\to\infty} \lambda\frac{\log M_n}{T_n} \geq R \tag{4}$$

$$\lim_{n\to\infty} \frac{n}{T_n} \geq \lambda \tag{5}$$

$$\lim_{n\to\infty} \epsilon_n \leq \epsilon \tag{6}$$

となるような符号の列 $\{(n, M_n, T_n, \epsilon_n)\}_{n=1}^{\infty}$ が存在することである．$0 < \epsilon < 1$ となる任意の ϵ において R が出力レート λ で ϵ-達成可能であれば，R は出力レート λ で達成可能であるという．出力レート λ で達成可能な R の上限を，出力レート λ における通信路容量と呼び，$C(\lambda)$ と表す．すなわち

$$C(\lambda) \triangleq \sup\{R|\ \text{符号化レート } R \text{ は出力レート } \lambda \text{ で達成可能.}\}$$

と定める．

[*1] 本論文を通して log は自然対数を表す．

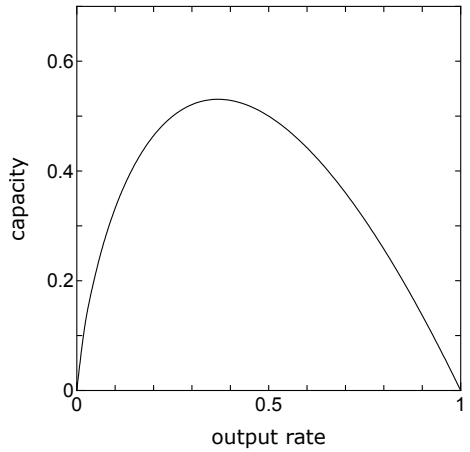

図 2　通信路容量

定義 2 と定義 3 には以下の関係がある．

定理 1 サービスレート μ のサーバを持つ通信路に対して C と $C(\lambda)$ は

$$C = \sup_{0<\lambda<\mu} C(\lambda) \tag{7}$$

を満たす [2].

定理 2　出力レート λ における通信路容量 $C(\lambda)$ は

$$C(\lambda) = \lambda \log \frac{\mu}{\lambda} \ \ (0 < \lambda < \mu) \tag{8}$$

と表される [1].

定理 2 を定理 1 に代入して最大値を求めると，通信路容量は

$$C = \frac{\mu}{e} \tag{9}$$

となる．そのときの出力レートは

$$\lambda = \frac{\mu}{e} \tag{10}$$

である．出力レート λ における通信路容量 $C(\lambda)$ のグラフを図 2 に示す．ただしサービスレートは $\mu = 1$ と固定した．

3　符号器と復号器の設計

本研究ではよい符号の列を作ること，すなわち符号器と復号器を設計することが目的である．本研究では一つの符号器とそれに対して 3 つの復号器を作り，それぞれの性能を比較し検

討する．

3.1 ポアソン過程による符号語の生成

文献 [1] では，出力レート λ における通信路容量 $C(\lambda)$ は通信路の入出力の相互情報量の上限を用いることで表されることが示されている．そのとき，相互情報量を最大化する入力過程はレート λ のポアソン過程である．したがって，通信路容量 C はレート $\lambda = \mu/e$ のポアソン過程と通信路の出力過程の間の相互情報量によって表される．本研究ではランダム符号化の技法を用い，レート $\lambda = \mu/e$ のポアソン過程によって符号語を生成し，平均的な符号の性能を調べることによって達成可能な符号化レートについて議論する．

符号語をレート $\lambda = \mu/e$ のポアソン過程によって生成することにしたので，符号化レートはパケット数と符号語数によって以下のように定まる．符号の符号化レートを R とおくと，定義 1 より

$$R = \frac{\log M}{T} \tag{11}$$

と書ける．一方，通信路の出力過程もレート λ のポアソン過程となるので，最後のパケットが到着する平均時刻 T は

$$T \simeq \frac{n}{\lambda} \tag{12}$$

となり[*2]，パケット数 n と $\lambda = \mu/e$ によって決まる．これを式 (11) に代入し，変形すると符号語数 M は

$$M \simeq e^{\frac{neR}{\mu}} \tag{13}$$

となる．これにより，目標の符号化レートを得るために，何個の符号語を用意すべきかがわかる．なお，符号語数は自然数なので実際には小数点以下を切り捨てる．

3.2 距離関数に基づく復号

復号器はサービス器から出力された n 個のパケットの出発時刻を観測する．そして，それらのパケットの時間間隔を受信語として受け取る．一般に，復号は復号領域に従って行われる．すなわち，各符号語に対する復号領域が存在し，受信語がその復号領域に入っていれば対応する符号語が復号される．本研究では復号領域を明示的に設計するのではなく，受信語と符号語の距離に基づいて復号を行う．具体的には，距離に基づいて受信語に一番「近い」符号語を復号語とする[*3]．

[*2] 式 (12) は $\lim_{n\to\infty} T/n = \lambda$ を意味する．

[*3] ここで慣例的に距離という用語を使ったが，実際には距離の公理を満たす必要はない．正確には距離ではなく歪みである．

以降，受信語を $b^n = (b_1, b_2, \cdots, b_n)$，符号語を $a^n = (a_1, a_2, \cdots, a_n)$ と表し，従来研究 [3] で用いられた 2 つの距離関数と，本研究で提案する 1 つの距離関数，合計 3 つの距離関数を導入する．

3.2.1 距離関数 1

一般的な AWGN 通信路 [4] における最尤復号則である 2 乗ユークリッド距離を計算し，送信された符号語を推定する．文献 [3] に倣い，受信語と符号語との距離を

$$d_n^{(1)}(b^n, a^n) \triangleq \sum_{i=1}^{n} (b_i - a_i)^2 \tag{14}$$

という距離関数で表す．

3.2.2 距離関数 2

パケットの時間間隔を用いて通信すると，符号語はキューとサービス器を通過するので，受信語の n 番目のパケットの受信時刻は，必ず送信された符号語の n 番目のパケットの送信時刻より後になる．すなわち，パケットの受信時刻が符号語の送信時刻と同じか，早くなることはありえない．文献 [3] ではこの単純な知見から，起こりえない符号語を復号されないように取り除く距離関数が定義されている．本研究ではこの距離関数を，距離関数 2 として用いる．

この距離の考え方を説明するために，パケットが一つの場合から考える．このとき，符号語と受信語は 1 つのパケットの到着時刻として表される．復号器は受信語 $b^1 = (b_1)$ を受け取り，符号器で生成された M 個の符号語のうちどの符号語が送信されたか推定する．このとき，時刻 b_1 のほうが時刻 a_1 より後の場合しかありえない．したがって $b_1 \leq a_1$ の場合は，起こりえない符号語が復号語とならないように，受信語と符号語の距離を ∞ とおく．$b_1 > a_1$ の場合は，時刻 b_1 と時刻 a_1 の差を距離とする．この距離が小さいということは，パケットの時間間隔が受信語と符号語で似ているということである．この考え方に基づき，パケットが 1 つの場合の距離関数を

$$d_1^{(2)}\left(b^1, a^1\right) \triangleq \begin{cases} b_1 - a_1 & (b_1 > a_1 \text{のとき}) \\ \infty & (b_1 \leq a_1 \text{のとき}) \end{cases} \tag{15}$$

と定める．

次にパケットが 2 つの場合を考えると，このとき受信語は $b^2 = (b_1, b_2)$ である．パケットが一つの場合と同様に，時刻 $b_1 + b_2$ が時刻 $a_1 + a_2$ より後の場合しかありえない．つまり，$b_1 + b_2 \leq a_1 + a_2$ は起こりえないので式 (15) と同様に復号されないようにする．この考え方

に基づき，パケットが 2 つの場合の距離関数を

$$d_2^{(2)}(b^2,a^2) \triangleq \begin{cases} d_1^{(2)}\left(b^1,a^1\right)+(b_1+b_2)-(a_1+a_2) & (b_1+b_2>a_1+a_2\text{のとき}) \\ \infty & (b_1+b_2\le a_1+a_2\text{のとき}) \end{cases} \tag{16}$$

と定める．なお，1 番目のパケットの時刻が $b_1 > a_1$ を満たしていなければ，距離 $d_2^{(2)}$ は ∞ とする．

一般にパケットが n 個の場合は，n 番目のパケットの到着時刻が起こりうる $\sum_{i=1}^{n} b_i > \sum_{i=1}^{n} a_i$ の場合と，起こりえない $\sum_{i=1}^{n-1} b_i \le \sum_{i=1}^{n} a_i$ の場合に分けて考える．式 (15) と (16) を考慮し，パケットが n 個の場合の距離関数を

$$d_n^{(2)}(b^n,a^n) \triangleq \begin{cases} d_{n-1}^{(2)}(b^{n-1},a^{n-1})+\sum_{i=1}^{n} b_i-\sum_{i=1}^{n} a_i & (\sum_{i=1}^{n} b_i>\sum_{i=1}^{n} a_i\text{のとき}) \\ \infty & (\sum_{i=1}^{n} b_i\le\sum_{i=1}^{n} a_i\text{のとき}) \end{cases} \tag{17}$$

と定める．こちらも $b_1 > a_1$，$b_1+b_2 > a_1+a_2$，$\cdots$，$\sum_{i=1}^{n-1} b_i > \sum_{i=1}^{n-1} a_i$ をすべて満たしていなければ，$d_n^{(2)}$ は ∞ となる．

3.2.3 距離関数 3

この距離関数 3 が，本研究で提案する距離関数である．距離関数 2 では n 番目のパケットの到着時刻が起こり得るかどうか，という観点から距離を決定していたが，到着したパケットがキューで待たされている可能性については一切考慮されていなかった．そこで，距離関数 3 では起こりえない符号語を取り除くことに加え，サービス器に到着したパケットがキューで待たされている場合の待ち時間も考慮した距離関数を定義する．

パケットが一つのとき，キューで次のパケットが待たされることはないため，このときの距離関数は式 (15) と同じとし，

$$d_1^{(3)}\left(b^1,a^1\right) \triangleq \begin{cases} b_1-a_1 & (b_1>a_1\text{のとき}) \\ \infty & (b_1\le a_1\text{のとき}) \end{cases} \tag{18}$$

と定める．

パケットが 2 つのときは場合分けをして考える必要がある．受信語 $b^2=(b_1,b_2)$ を受信したとする．このとき，2 番目のパケットがキューで待たない $b_1 \le a_1+a_2$ の場合を図 3(a) に，2 番目のパケットがキューで待つ $b_1 > a_1+a_2$ の場合を図 3(b) に示す．

図 3 はパケットの通信路への到着，そして通信路を介してからの出発の様子を時間軸上で表したものである．下から時間軸への矢印はパケットの到着時刻，時間軸から上への矢印はパケットの出発時刻を表している．そして横向きの矢印で示した時間が，距離関数 3 で用いる距離である．この場合分けを用いることで，距離関数 2 と比べて，受信語と送信したと思われる

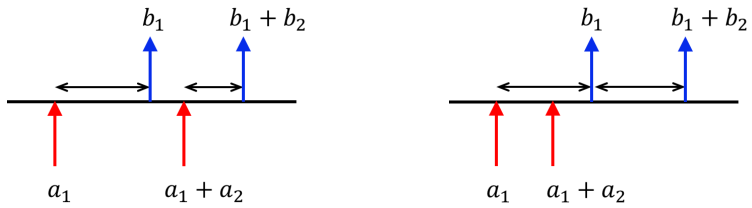

図 3　パケットが 2 つのときの場合分け

符号語の距離をより「近く」設計できると考えた．この考え方に基づき，パケットが 2 つの場合の距離関数を

$$d_2^{(3)}(b^2, a^2) \triangleq \begin{cases} d_1^{(3)}(b^1, a^1) + (b_1 + b_2) - (a_1 + a_2) & (b_1 \le a_1 + a_2 \text{のとき}) \\ d_1^{(3)}(b^1, a^1) + b_2 & (b_1 > a_1 + a_2 \text{のとき}) \end{cases} \tag{19}$$

と定める．なお，起こりえない符号語は取り除かれるように $b_1 > a_1$ かつ $b_1 + b_2 > a_1 + a_2$ を満たしていなければ，$d_2^{(3)}$ は ∞ とする．ここで，$b_1 > a_1$ かつ $b_1 + b_2 > a_1 + a_2$ を満たしている符号語，すなわち起こりうる符号語との距離のみを考えれば，式 (19) は

$$d_2^{(3)}(b^2, a^2) = (b_1 - a_1) + (b_1 + b_2) - \max\{b_1, a_1 + a_2\} \tag{20}$$

と書き換えることができる．

一般にパケットが n 個の場合は，n 番目のパケットがキューで待たない $\sum_{i=1}^{n-1} b_i \le \sum_{i=1}^{n} a_i$ の場合と，n 番目のパケットがキューで待つ $\sum_{i=1}^{n-1} b_i > \sum_{i=1}^{n} a_i$ の場合を考える．式 (18) と (19) を考慮した上で，パケットが n 個の場合の距離関数を

$$d_n^{(3)}(b^n, a^n) \triangleq \begin{cases} d_{n-1}^{(3)}(b^{n-1}, a^{n-1}) + \sum_{i=1}^{n} b_i - \sum_{i=1}^{n} a_i & (\sum_{i=1}^{n-1} b_i \le \sum_{i=1}^{n} a_i \text{のとき}) \\ d_{n-1}^{(3)}(b^{n-1}, a^{n-1}) + b_n & (\sum_{i=1}^{n-1} b_i > \sum_{i=1}^{n} a_i \text{のとき}) \end{cases} \tag{21}$$

と定める．こちらも，$b_1 > a_1$，$b_1 + b_2 > a_1 + a_2$，$\cdots$，$\sum_{i=1}^{n} b_i > \sum_{i=1}^{n} a_i$ をすべて満たしていなければ，$d_n^{(3)}$ は ∞ とする．式 (20) と同様に，起こりうる符号語との距離のみを考えれば，式 (21) は

$$\begin{aligned} & d_n^{(3)}(b^n, a^n) \\ & = b_1 - a_1 + (b_1 + b_2) - \max\{b_1, a_1 + a_2\} \\ & \qquad + (b_1 + b_2 + b_3) - \max\{b_1 + b_2, a_1 + a_2 + a_3\} \\ & \qquad + \cdots \\ & \qquad + (b_1 + b_2 + \cdots + b_n) - \max\{b_1 + b_2 + \cdots + b_{n-1}, a_1 + a_2 + \cdots + a_n\} \end{aligned} \tag{22}$$

と書き換えることができる．ここで，

$$\hat{a}_i \triangleq \sum_{k=1}^{i} a_k$$
$$\hat{b}_i \triangleq \sum_{k=1}^{i} b_k$$

とおくことで，式 (22) は

$$d_n^{(3)}(b^n, a^n) = b_1 - a_1 + \sum_{i=2}^{n} \left(\hat{b}_i - \max\{\hat{b}_{i-1}, \hat{a}_i\} \right) \tag{23}$$

と書ける．

3.3 最尤復号のための距離関数

じつのところ，すべての符号語の生起確率が一定であるとき，距離関数 3 を用いた復号は本研究で扱う通信路において最尤復号法になっている．これは，サービス時間がサービスレート μ の指数分布に従って，確率的に決定するということに大きく起因している．この通信路に符号語 $a^n = (a_1, a_2, \cdots, a_n)$ を送信し，受信語 $b^n = (b_1, b_2, \cdots, b_n)$ を受信したとして，この最尤復号について説明する．

まず初めに，1 つ目のパケットのサービス時間はキューが空であることから必ず $b_1 - a_1$ となる．指数分布の確率密度関数を考えると，サービス時間が $b_1 - a_1$ になる確率密度は

$$\Pr\{ \text{サービス時間が } b_1 - a_1 \} = \mu \exp\{-\mu(b_1 - a_1)\} \tag{24}$$

で与えられる．

続いて 2 つ目のパケットのサービス時間であるが，これは図 3 に示したように，2 通りのサービス時間が考えられえる．2 つ目のパケットがキューでの待ち時間がない $b_1 \leq a_1 + a_2$ の場合と 2 つ目のパケットがキューでの待ち時間を要する $b_1 > a_1 + a_2$ の場合である．各サービス時間はパケットの到着時刻と出発時刻に依存せず互いに独立であるため，$b_1 \leq a_1 + a_2$ のとき

$$\begin{aligned}
&\Pr\{ \text{1 番目のサービス時間が } b_1 - a_1 \text{ かつ 2 番目のサービス時間が } (b_1 + b_2) - (a_1 + a_2) \ \} \\
&= \Pr\{\text{1 番目のサービス時間が } b_1 - a_1\} \Pr\{\text{2 番目のサービス時間が } (b_1 + b_2) - (a_1 + a_2)\} \\
&= \mu \exp\{-\mu(b_1 - a_1)\} \mu \exp\{-\mu((b_1 + b_2) - (a_1 + a_2))\} \\
&= \mu^2 \exp\{-\mu((b_1 - a_1) + (b_1 + b_2) - (a_1 + a_2))\}
\end{aligned} \tag{25}$$

となり，$b_1 > a_1 + a_2$ のときは

$$\begin{aligned}
&\Pr\{1\text{ 番目のサービス時間が }b_1 - a_1\text{ かつ }2\text{ 番目のサービス時間が }b_2\} \\
&= \Pr\{1\text{ 番目のサービス時間が }b_1 - a_1\}\Pr\{2\text{ 番目のサービス時間が }b_2\} \\
&= \mu\exp\{-\mu(b_1 - a_1)\}\mu\exp\{-\mu b_2\} \\
&= \mu^2\exp\{-\mu((b_1 - a_1) + b_2)\} \qquad (26)
\end{aligned}$$

となる．ここで式 (25)，(26) は，まさに符号語 $a^2 = (a_1, a_2)$ を送信したというもとで，受信語 $b^2 = (b_1, b_2)$ を受信する確率密度を表している．これを条件付き確率としてまとめると

$$\begin{aligned}
&\Pr\{\text{ 受信語が }b^2 \mid \text{符号語が }a^2\} \\
&= \mu^2\exp\{-\mu((b_1 - a_1) + (b_1 + b_2) - \max\{a_1 + a_2, b_1\})\} \qquad (27)
\end{aligned}$$

と書くことができる．

一般に n 番目のパケットのサービス時間は，$\sum_{i=1}^{n-1} b_i \le \sum_{i=1}^{n} a_i$ のとき $\sum_{i=1}^{n} b_i - \sum_{i=1}^{n} a_i$，$\sum_{i=1}^{n-1} b_i > \sum_{i=1}^{n} a_i$ のとき b_n となる．これより，符号語 $a^n = (a_1, a_2, \cdots, a_n)$ を送信したというもとで，受信語 $b^n = (b_1, b_2, \cdots, b_n)$ を受信する確率密度は

$$\begin{aligned}
&\Pr\{\text{ 受信語が }b^n \mid \text{符号語が }a^n\} \\
&= \mu^n\exp\{-\mu((b_1 - a_1) \\
&\qquad + (b_1 + b_2) - \max\{a_1 + a_2, b_1\} \\
&\qquad + (b_1 + b_2 + b_3) - \max\{b_1 + b_2, a_1 + a_2 + a_3\} \\
&\qquad + \cdots \\
&\qquad + (b_1 + b_2 + \cdots + b_n) - \max\{b_1 + b_2 + \cdots + b_{n-1}, a_1 + a_2 + \cdots + a_n\})\} \qquad (28)
\end{aligned}$$

と書くことができ，

$$\begin{aligned}
\hat{a}_i &\triangleq \sum_{k=1}^{i} a_k \\
\hat{b}_i &\triangleq \sum_{k=1}^{i} b_k
\end{aligned}$$

とおくことで，式 (28) は

$$\begin{aligned}
\Pr\{\text{ 受信語が }b^n \mid \text{符号語が }a^n\} &= \mu^n\exp\left\{-\mu\left(b_1 - a_1 + \sum_{i=2}^{n}\left(\hat{b_i} - \max\{\hat{b}_{i-1}, \hat{a}_i\}\right)\right)\right\} \\
&= \mu^n\exp\left\{-\mu\left(d_n^{(3)}(b^n, a^n)\right)\right\} \qquad (29)
\end{aligned}$$

と $d_n^{(3)}$ を用いて書ける．

メッセージが等確率で発生するとき，条件付き確率は最尤復号のための尤度関数なので，距離 $d_n^{(3)}$ が最も小さい符号語は尤度関数の値を最大化させる．よってこの距離関数 3 に基づく復号法は本研究で扱う通信路において最尤復号法となる．

4　復号器の評価

4.1　評価方法

本研究では，2.1 節で述べたとおり，サービス器のサービスレートは $\mu = 1$ として復号器の評価を行う．このとき，出力レート $\lambda = e^{-1} = 0.367$[symbol/sec] で通信路容量 $C = 0.367$[symbol/sec] を達成できる．λ の値を 0.367 に固定し，3.1 節で述べた方法で目標の符号化レートを得るための符号語数を決定する．M 個の符号語から 1 つの符号語を等確率で選んで送信し，3 章で用意した 3 種類の距離関数によって復号するシミュレーションを 100 万回行った．このとき，パケット数 n を増加させたときの復号誤り確率 ϵ を計測した．パケット数 n の増加に伴い，復号誤り確率 ϵ が 0 に近づいていく傾向があるかどうかで，その時の符号化レートが達成可能かどうか判断した．

4.2　結果と考察

はじめに，符号化レート R の値が通信路容量と等しい $R = 0.367$ のときの結果を図 4 に示す．横軸にパケット数 n，縦軸に対数スケールで復号誤り確率 ϵ をプロットした．このとき，

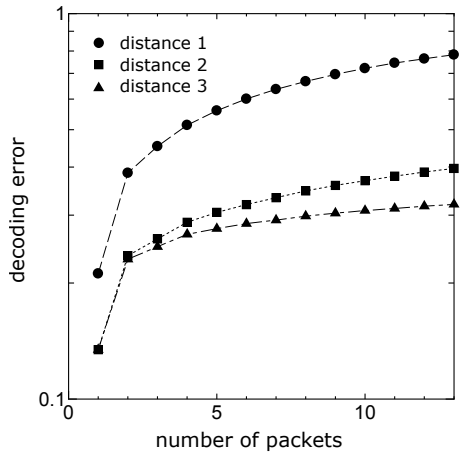

図 4　$R = 0.367$ のときの復号誤り確率

3 つの距離関数とも n の増加とともに復号誤り率も増加してゆくことがわかる．距離関数 3

(distance 3) では最尤復号をしているにもかかわらず，復号誤り確率が減少していかない．これは n を十分大きな値まで増加させることができなかったからだと考えられる．

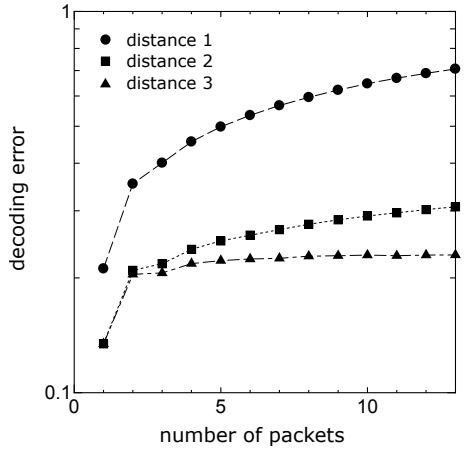

図 5　$R = 0.330$ のときの復号誤り確率

次に，R を通信路容量の 90% に減少させた $R = 0.330$ のときの結果を図 5 に示す．このときも，3 つの距離関数とも n の増加とともに復号誤り率も増加している．距離関数 3 (distance 3) の誤り確率は緩やかではあるが上昇している．

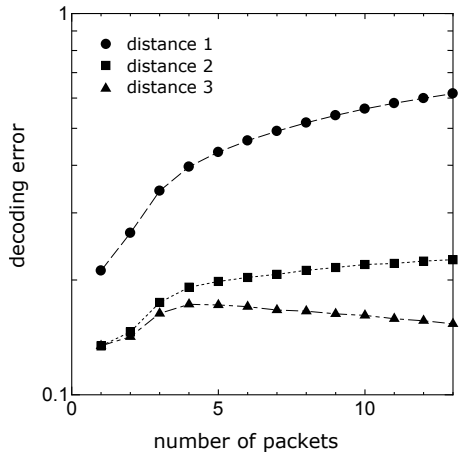

図 6　$R = 0.294$ のときの復号誤り確率

次に通信路容量の 80% に減少させた $R = 0.294$ のときの結果を図 6 に示す．このときは距離関数 3 (distance 3) のみ，n の増加に伴い，復号誤り確率が指数関数的に減少していくこと

が見て取れる．このことから，n を十分大きくしたとき，復号誤り確率が極限で 0 にいくと予想できる．このため，距離関数 3 による復号で $R = 0.294$ を達成できると言える．なお，距離関数 1（distance 1）と距離関数 2（distance 2）の復号誤り確率はいまだに上昇している．

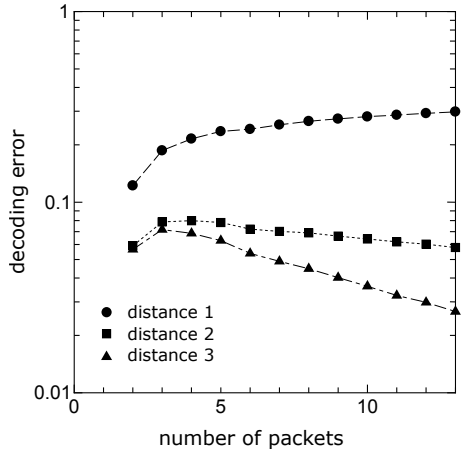

図 7　$R = 0.184$ のときの復号誤り確率

次に通信路容量の 50% に減少させた $R = 0.257$ のときの結果を図 7 に示す．符号化レートをここまで下げると，距離関数 2（distance 2）の復号誤り確率も n の増加に伴い減少していくことが見て取れる．このため，距離関数 2（distance 2）による復号で $R = 0.184$ を達成できると言える．なお，距離関数 1（distance 1）の復号誤り確率はいまだに上昇している．

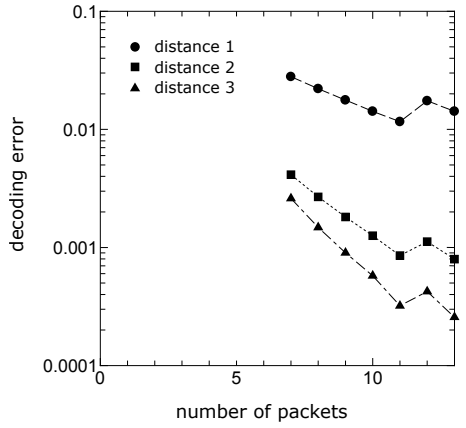

図 8　$R = 0.0367$ のときの復号誤り確率

最後に通信路容量の 10% に減少させた $R = 0.0367$ のときの結果を図 8 に示す．ようやく距離関数 1（distance 1）の復号誤り確率も n の増加に伴い減少していくことが見て取れる．

5 まとめ

通信手段の一つとして通信路に送られるパケットの時間間隔に着目し，その間隔によって情報を送ることができると文献 [1] で指摘されている．同時に図 1 のような通信システムで時間間隔を情報として送受信した場合の通信路容量も求められている．

本研究では通信路容量を達成する符号の構成を，復号器の設計という観点から考えた．文献 [2] に倣った距離関数 1 と距離関数 2 に基づく復号法に加え，距離関数 2 の考え方をより精緻した距離関数 3 を提案した．さらに，距離関数 3 に基づく復号は，本研究で扱う通信路において最尤復号になっていることを証明した．シミュレーションによる誤り確率の計測の結果を見ると，距離関数 1 が達成可能な符号化レートの上限はおよそ $R = 0.0367$，距離関数 2 ではおよそ $R = 0.184$，距離関数 3 ではおよそ $R = 0.294$ であった．距離関数 3 が最尤復号法であるにも関わらず，シミュレーションで通信路容量を達成できないであろう結果が得られたのは，n の値を 13 以上に大きくすることができなかったからだと考えられる．このとき，n を十分大きな値でシミュレーションすると，どのような結果が得られるかとても興味深い．

今後の課題としては，パケット間隔を用いた符号が現実社会でどのように実装されるべきか，探っていきたい．距離関数 3 による復号法は理論的には最適である．しかしながら，この符号を我々の日常生活で活用させるには，符号器・復号器を実装するコスト，規模等を考えなければならない．こういった実際問題に焦点を当てた研究も行っていく必要があるといえる．

謝辞

本研究を行うにあたり，数多くの助言，指導をしてくださった指導教員西新幹彦准教授，また西新研究室の皆様に感謝の意を表する．

参考文献

[1] Venkat Anantharam, Sergio Verdú, ”Bits Through Queues,” IEEE Transactions on Information Theory, vol.42, no.1, pp.4-18, Jan. 1996.

[2] 西新幹彦，藤山直貴，「パケット間隔で情報を送る通信路の悲観的な符号化について」，第35 回情報理論とその応用シンポジウム (SITA2012)，pp.590-595, Dec. 2012.

[3] Abdullah Fahim, 「パケット間隔で情報を送るシステムのための復号法に関する検討」，信州大学工学部，学士論文，2014.

[4] 和田山正，誤り訂正技術の基礎，森北出版，2010.

付録 A　ソースコード

A.1　符号語をパケット数に応じた個数だけ生成し復号誤り回数を計測するプログラム

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

#define MRND 1000000000L
static int jrand;
static long ia[56];

//一 様乱数の生成
static void irn55(void)
{
        int i;
        long j;

        for (i = 1; i <= 24; i++){
                j = ia[i] - ia[i + 31];
                if (j < 0) j += MRND;
                ia[i] = j;
        }
        for (i = 25; i <= 55; i++){
                j = ia[i] - ia[i - 24];
                if (j < 0) j += MRND;
                ia[i] = j;
        }
}

void init_rnd(unsigned long seed)
{
        int i, ii;
        long k;
        ia[55] = seed;
        k = 1;

        for (i = 1; i <= 54; i++){
                ii = (21 * i) % 55;
                ia[ii] = k;
                k = seed - k;
                if (k<0) k += MRND;
                seed = ia[ii];
        }
        irn55(); irn55(); irn55();
        jrand = 55;
}

long irnd(void)
{
        if (++jrand>55){ irn55(); jrand = 1; }
        return ia[jrand];
}

double rnd(void)
{
        return (1.0 / MRND)*irnd();
}

//一様分布→ rate=1 の指数分布への変換
```

```
double rand_e(void)
{
        return (-log(1 - rnd()));
}

//rate=0.367 の指数分布へ変換
double rand_e2(void)
{
        return ((rand_e())/0.367);
}

//送受信
void
transmit(int n, double snd[], double rcv[])
{
        int i = 0;
        int j = 0;
        double current_time = 0;
        double service_end = 0;
        int customer_num = 0;

        while (i < n || customer_num > 0){
                printf("customer: %d, i: %d\n", customer_num, i);
                if (customer_num == 0 || (i < n && snd[i] < service_end)){
                        //到着が先
                        printf("到着した (%f)\n", snd[i]);
                        current_time = snd[i];
                        i++;
                        if (customer_num >= 1){
                                customer_num++;
                        }
                        else {
                                service_end = current_time + rand_e();
                                customer_num = 1;
                        }
                }
                else {
                        //サービス終了
                        printf("サービス終了した (%f)\n", service_end);
                        current_time = service_end;
                        rcv[j] = service_end;
                        j++;
                        customer_num--;
                        if (customer_num >= 1){
                                service_end = current_time + rand_e();
                        }
                }
        }
        return;
}

//大小比較
double max(double snd, double rcv)
{
        double a;
        if(snd > rcv){
                a = snd;
        }
        else{
                a = rcv;
        }
        return(a);
}



//距離関数 1
double
dist0(int n, double snd[], double rcv[])
```

```
{
        double dist = 0.0;
        double dif, dif0, dif1;
        int i;

        dif0 = 0.0;
        for (i = 0; i < n; i++){
                dif1 = snd[i] - rcv[i];
                dif = dif1 -  dif0;
                dist += dif * dif;
                dif0 = dif1;
        }
        return(1.0 / dist);
}

//距離関数 2
double
dist1(int n, double snd[], double rcv[])
{
        double dist = 0.0;
        int i;

        for (i = 0; i < n; i++){
                if (snd[i] > rcv[i]){
                        return(0);
                } else {
                        double dif = rcv[i] - snd[i];
                    dist += dif;
                }
        }
        return(1.0 / dist);
}

//距離関数 3
double
dist2(int n, double snd[], double rcv[])
{
        double dist = rcv[0] - snd[0];
        int i;

        if(snd[0] > rcv[0]){
                return(0);
        }
        else{
                for(i = 1; i < n; i++){
                        if(snd[i] > rcv[i]){
                                return(0);
                        }
                        else{
                                dist += rcv[i] - max(rcv[i-1], snd[i]);
                        }
                }
        }
        if(dist == 0){
                printf("dist = 0\n");
                return(DBL_MAX);
        }
        return(1.0 / dist);
}

//誤る回数 k をはかる
void decording(int n, int k_max, int k[3] )
{
        double cordword[20];
        double rcv[20];
        double dist[3];
        int i;
        int j;
```

```
//オリジナル符号語作成
cordword[0] = rand_e2();
for(i = 1; i < n; i++){
        cordword[i] = cordword[i-1] + rand_e2();
}

for (i = 0; i < n; i++){
        printf("オリジナル符号語：%f\n", cordword[i]);
}


transmit(n, cordword, rcv);

for (i = 0; i < n; i++){
        printf("受信語：%f\n", rcv[i]);
}

//オリジナルの dist をはかる
dist[0] = dist0(n, cordword, rcv);
dist[1] = dist1(n, cordword, rcv);
dist[2] = dist2(n, cordword, rcv);
printf("符号語との尤度: %f, %f, %f\n", dist[0], dist[1], dist[2]);

k[0] = k[1] = k[2] = 0;

for(i = 1; i < k_max; i++){
        double d;

        //ダミー符号語作成
        cordword[0] = rand_e2();
        for(j = 1; j < n; j++){
                cordword[j] = cordword[j-1] + rand_e2();
        }
        //オリジナルの dist と比較
        if (k[0] == 0){
                d = dist0(n, cordword, rcv);
            printf("ダミーとの尤度: %f, ", d);
                if (d > dist[0]){
                        k[0] = i;
                }
        }
        if (k[1] == 0){
                d = dist1(n, cordword, rcv);
            printf("%f, ", d);
                if (d > dist[1]){
                        k[1] = i;
                }
        }
        if (k[2] == 0){
                d = dist2(n, cordword, rcv);
          printf("%f\n", d);
                if (d > dist[2]){
                        k[2] = i;
                }
        }

        if(k[0] > 0 && k[1] > 0 && k[2] > 0){
                break;
        }
}
if(k[0] == 0){
        k[0] = k_max;
}
if(k[1] == 0){
        k[1] = k_max;
}
if(k[2] == 0){
        k[2] = k_max;
}
```

```
        return;
}

int
main(int argc, char **argv)
{
        init_rnd(1969);
        int i;
        int n;
        int rep;
        int k[3];
        char filename[128];

        if (argc < 3){
                printf("使用法: ファイル名ベース パケット数\n");
                exit(1);
        }

        n = atoi(argv[2]);
        //ファイルへの出力
        sprintf(filename, "%s%d.txt", argv[1], n);
    FILE *fp;
    if((fp = fopen(filename, "a")) == NULL) {
                printf("出力ファイルをオープンできません.\n");
                exit(1);
        }
        printf("%s に結果を保存しています (n = %d)\n", filename, n);

printf("dist0: %f\n", dist0(3, snd, rcv));
printf("dist1: %f\n", dist1(3, snd, rcv));
printf("dist2: %f\n", dist2(3, snd, rcv));

#define LOOP 1000000
        rep = exp((double)n) + 1.0;
        for (i = 0; i < LOOP; i++){
                decording(n, rep, k);
                fprintf(fp,"%d,%d,%d\n",k[0],k[1],k[2]);

                if (i % (LOOP / 100000) == 0){
                        printf("\r %.3f%%", i * 100.0 / LOOP);
                }
printf("k[0]: %d\n", k[0]);
printf("k[1]: %d\n", k[1]);
printf("k[2]: %d\n", k[2]);
        }

        fclose(fp);

        return(0);
}
```

A.2 符号化レート R を入力した際に各パケット数 n に対する復号誤り確率 ϵ を返すプログラム

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

#define LOOP 1000000

double coding_rate = 0.367;

void
get_error_rate(int n, double error_rate[])
{
        int dist[3];
        int error_count[3];
        char filename[128];
    FILE *fp;
        int i;
        int mn;

        sprintf(filename, "data%d.txt", n);
        //ファイルからの入力
        if((fp = fopen(filename, "r")) == NULL) {
                printf("ファイル %s をオープンできません.\n", filename);
                exit(1);
        }

        mn = (int)exp(n * coding_rate * 2.7182818);
        error_count[0] = error_count[1] = error_count[2] = 0;
        for(i = 0; i < LOOP; i++){
                if(fscanf(fp,"%d,%d,%d", dist, dist + 1, dist + 2) < 3){
                        printf("データファイル %s にデータが足りません。\n", filename);
                        exit(1);
                }
                if(dist[0] < mn){
                        error_count[0]++;
                }
                if(dist[1] < mn){
                        error_count[1]++;
                }
                if(dist[2] < mn){
                        error_count[2]++;
                }
        }
        fclose(fp);
        error_rate[0] = (double)error_count[0] / LOOP;
        error_rate[1] = (double)error_count[1] / LOOP;
        error_rate[2] = (double)error_count[2] / LOOP;
        return;
}

int
main(int argc, char **argv)
{
        int n; /* パケット数 */
        double error_rate[3];

        if (argc < 2){
                printf("使用法: 符号化レート\n");
                exit(1);
```

```
        }

        coding_rate = atof(argv[1]);

        printf("coding_rate = %f\n",coding_rate);
        printf("パケット数\t 尤度 0\t 尤度 1\t 尤度 2\n");
        for (n = 1; n < 14; n++){
                printf("%d\t", n);
                get_error_rate(n, error_rate);
                printf("%f\t%f\t%f\n", error_rate[0], error_rate[1], error_rate[2]);
        }

        return(0);
}
```