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 ランダム符号化における符号語の生成確率の影響

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　14T2008C
氏名　市川　謙吾

2018 年 2 月 21 日

目次

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ー通信において，データを小さなまとまりに分割してその一つ一つを送受信するという通信方式である．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使用することができる．また通信経路の選択をすることができるのでネットワーク上で一部に障害が生じてしまったとしてもほかの回線を利用し通信を行う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が，Anantharam and Verdú[1] によると, パケット間隔も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さに意味を与えておくことで受信者は情報を受け取ることができる．パケット通信を行う際は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移動などが発生してしまい，パケットの時間間隔が送信時と受信時とで異なってしまうため復号器での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与えた時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いまだ知られておらず，現実的な符号の構成に向けていくつかの研究がなされている．例えば従来研究 [3] では通信路容量の定義を踏まえて，符号語長を大きくしていった際の符号語長と復号誤り確率の関係に基づき，尤度関数を検証している．本研究では従来研究により検証された尤度関数を用いて，ランダム符号化の符号の生成確率が復号誤り確率に与える影響を実験的に検証する．本研究の結果，ランダム符号化の符号の生成確率は，理論的な最適値からずらしても誤り確率を小さくできないことが分かった．

1.2　本論文の構成

本論文は次のような章から構成される．2 章では本研究で扱う通信システムの説明，符号の定義，そして，出力レート，通信路容量について述べる．3 章ではランダム符号化による復号誤り確率の測定方法とその測定の際に使用する尤度関数について説明する．4 章ではランダム符号化の生成確率が復号誤り確率に与える影響を検証する．最後に 5 章でまとめを述べる．

2　通信システム

この章では本研究の通信路モデルと符号の形式を説明し，既知の結果である通信路符号化定理を述べる．

2.1　通信路モデル

本研究では，従来研究 [3] と同じ通信システムについて検討する．そのため，通信システムの説明の多くは従来研究 [3] からの引用である．

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パケットを受信者に不規則に送信する．そしてパケットは複数のサーバを通り最終的な目的地に到着する．パケットが送信者から出発してから目的地に到着するまでの時間は，ほかのユーザが送信しているパケットの数，すなわちネットワーク上での混み具合などに依存する．そして一組の送受信者に着目して考えるとネットワーク全体は複雑な待ち行列システムとして表される．本研究では簡単化のため複雑な待ち行列システムをシンプルな単一サーバ FIFO 待ち行列システムとみなす．そのため，サービスを受けている最中に到着したパケットの順序を崩さないためにサーバの前に十分大きなキューを設置する．このような待ち行列システムによる通信路モデルを図 1 に示す．

さらに，パケットの中に情報を入れて通信するのが普通のパケット通信であるが，本研究ではパケットには情報を入れず，パケットの送信間隔のみで通信を行う．すなわち，符号器では，送信されるメッセージを時間間隔の列に変換し通信路へと送信する．サーバ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．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．符号器と復号器のペアを符号と呼ぶ．

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り，本研究では従来研究 [1,3] に従い，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　符号語は 非負実数ベクトル $(a_1, a_2, \cdots, a_n)$ として表される．符号器が符号語

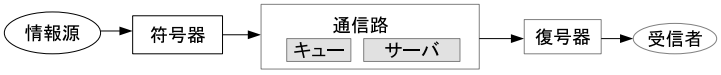

図 1　通信路モデル

$(a_1, a_2, \cdots, a_n)$ 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．復号器にとっての受信語とは，サーバからの n 個の出発間隔である．復号器の復号誤り確率を ϵ とおく．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．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．なお本論文では log は自然対数を表す．

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ければ大きい程良いが，復号器で正しく復号されなければ意味がなくなってしまう．したがって与えられた符号化レートに対して正しく復号できるかが重要である．この概念について以下のように定義する [2].

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

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

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

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

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

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

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．$0 < \epsilon < 1$ 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

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

と定める.

定義 2 と定義 3 には以下のような関係がある.

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.

定理 2　出力レート λ $(0 < \lambda < \mu)$ における通信路容量 $C(\lambda)$ は

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

と表される.

定理 2 を定理 1 に代入し，最大値を求めると，通信路容量は

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

であり、その時の出力レートは

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

となる.

3　復号誤り確率の測定方法とその検証

本研究では，これまで述べた通信路に対してランダム符号化を用いて復号誤り確率を実験的に測定した. 本章では，測定方法を説明し，追実験により本測定が正しいとの確認が出来たことを報告する.

3.1　復号誤り確率の測定方法

本研究ではランダム符号化と尤度関数に基づく最尤復号で復号誤り確率を測定した．長さ n の符号語を作る際には同一パラメータの指数分布で n 個の独立な非負実数を発生させる．以降は，符号語を生成する際の指数分布のパラメータを生成レートと呼ぶこととする．符号化レート R の符号を構成するためには，原理的には e^{nR} 個の符号語を用意すればよい．しかし，本研究の測定手順では e^{nR} 個の符号語はあらかじめ用意せず測定を行った．具体的には次のような手順で行った.

復号誤り確率の測定は複数の試行からなっている．1 回の試行では，まず 1 つ目の符号語をランダムに生成する．次に，この符号語を送信したとみなし，通信路の影響をシミュレーショ

ンして復号器の受信語を生成する．そして符号語と受信語の尤度を測っておく．ここで一般に，符号語と受信語の尤度を単にその符号語の尤度と呼ぶことにする．次に，2 番目以降の符号語を次々にランダムに生成し，それぞれ尤度を測る．もし，2 番目から e^{nR} 番目までの符号語の中に 1 番目の符号語より尤度の大きいものが存在しなければ，この試行では符号語が正しく復号されたことを意味する．もし，そうでなければこの試行では復号誤りが発生したことを意味する．なお，符号語の生成の過程で 1 つでも尤度の大きいものが現れたら，残りの符号語を生成することなく復号誤りの発生を断定できる．このような試行を複数回行い，復号誤りが起きた割合を求めることによって復号誤り確率の測定値とした．

3.2 距離関数に基づく復号

前節で尤度関数に基づく最尤復号を行うことを述べたが，具体的な尤度関数は従来研究 [3] に従って以下に述べる 3 種類を使用した．実際には，各尤度関数は距離関数の逆数として定義される．以下にそれらの 3 つの距離関数の定義を述べる．以降，受信語を $b^n = (b_1, b_2, \cdots, b_n)$，符号語を $a^n = (a_1, a_2, \cdots, a_n)$ と表す．

距離関数 1 では受信語と符号語との距離を

$$d_n^{(1)}(b^n, a^n) \triangleq \sum_{i=1}^{n} (b_i - a_i)^2 \tag{11}$$

と定める．

距離関数 2 では場合分けが必要であり，パケットが 1 つの時の距離関数を

$$d_1^{(2)}(b^1, a^1) \triangleq \begin{cases} b_1 - a_1 & (b_1 > a_1 \text{の時}) \\ \infty & (b_1 \leq a_1 \text{の時}) \end{cases} \tag{12}$$

と定める．パケットが n 個の場合は一般に，n 番目のパケットの到着時刻が起こりうる場合と起こりえない場合とで分けて考える．距離関数は

$$d_n^{(2)} \triangleq \begin{cases} d_{n-1}^{(2)}(b^{n-1}, a^{n-1}) + \sum_{i=1}^{n} b_i - \sum_{i=1}^{n} a_i & (\sum_{i=1}^{n} b_i > \sum_{i=1}^{n} a_i \text{の時}) \\ \infty & (\sum_{i=1}^{n} b_i \leq \sum_{i=1}^{n} a_1 \text{の時}) \end{cases} \tag{13}$$

という距離関数で表す．

距離関数 3 は

$$d_n^{(3)}(b^n, a^n) = b_1 - a_1 + \sum_{i=2}^{n} \left(\hat{b_i} - max(\hat{b}_{i-1}, \hat{a_i}) \right) \tag{14}$$

という距離関数で表すことができる．

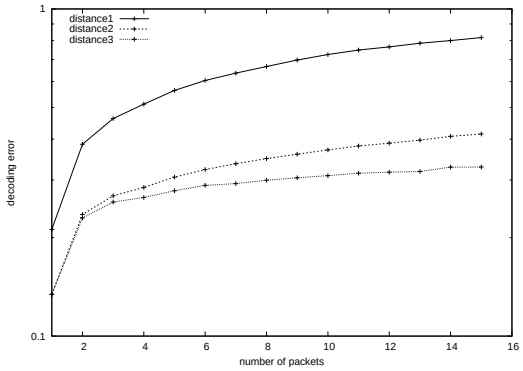

図 2　R=0.367 の時の復号誤り確率

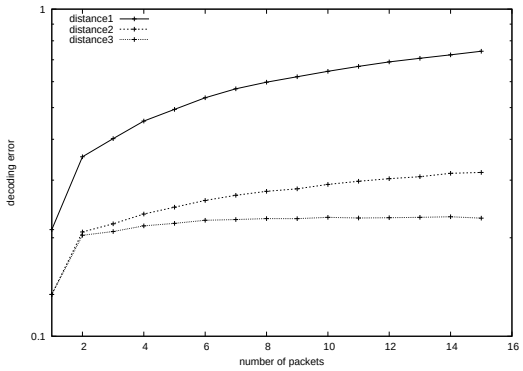

図 3　R=0.330 の時の復号誤り確率

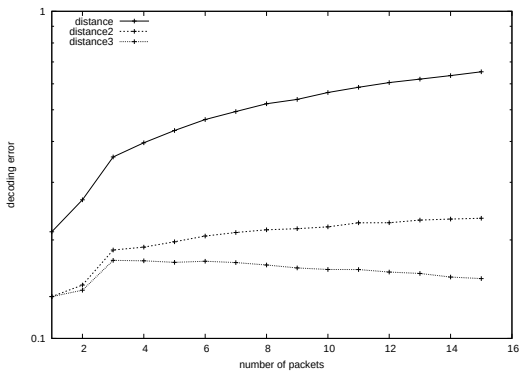

図 4　R=0.294 の時の復号誤り確率

3.3 測定方法の検証

測定方法の検証のために文献 [3] の追実験を行った. 追実験では，サーバのサービスレートは一般性を損なうことなく $\mu = 1$ として復号器の評価を行う．出力レート $\lambda = 0.367$[symbol/sec] で通信路容量 $C = 0.367$[nat/sec] を達成できることは従来研究 [1] より明らかにされている．生成レートは 0.367[symbol/sec] で固定し，ランダム符号化により符号語を生成し，前節で示した 3 種類の距離関数を用いて復号誤り確率を計測した．復号誤り確率一つあたり 100 万回の試行を行った．さらに，符号語長 n と復号誤り確率の関係を調べ，従来研究の結果と比較することで測定手順の検証を行った．

始めに符号化レート R の値が通信路容量と等しい R=0.367 の時の結果を図 2 に示す．横軸にパケット数，縦軸に対数スケールで復号誤り確率を取った．

次に符号化レートを R=0.330 の時の結果を図 3 に，符号化レートを R=0.294 の時の結果を図 4 に示す．図 2，図 3，図 4 のそれぞれのグラフより従来研究 [3] と同じ結果が得られ，本研究で作成したプログラムが正しく動作していることが検証できた．

4 最適な符号語生成確率の検証

この章では，符号語の生成確率と復号誤り確率の関係を知るために行った検証について述べていく．

4.1 背景，検証方法

従来研究 [1] では 2.3 節で紹介した出力レート，通信路容量を数学的に導き出している．しかし，この通信路容量の値を達成するための符号化法は具体的には示されていない．これに対し従来研究 [3] では，具体的な符号化法を構成するために，最尤復号法の尤度関数について考察している. その際，3.3 節で述べたようなランダム符号化による復号誤り確率の測定を行っており，符号語の生成確率として式 (10) に基づいた理論的に最適な生成レートを用いている. しかし，式 (10) は符号語長を限りなく大きくした場合に最適な値であって，有限長の符号語に対して最適かどうかは明らかではない. そこで本研究では，比較的短い符号語に対して最適な符号語の生成確率がどのようになっているかを実験的に検証する. この検証のため，ランダム符号化を用いて従来研究と同様の方法で，符号化レートの値は固定したうえで，符号語長 n を増やしていき，復号誤り確率を測定した．なお、この検証では従来研究により最適な尤度関数と示された距離関数 3 を用いて尤度を測定し，サービスレートは一般性を損なうことなく $\mu = 1$ で測定をした．

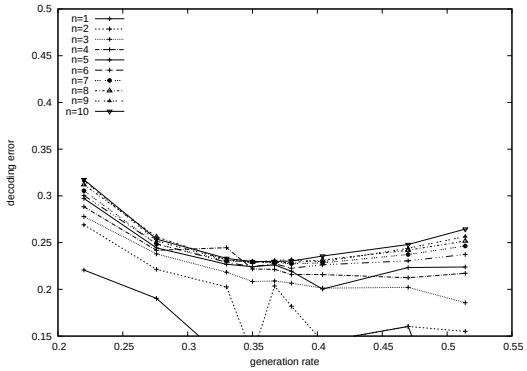

図 5　R=0.330 の時の復号誤り確率

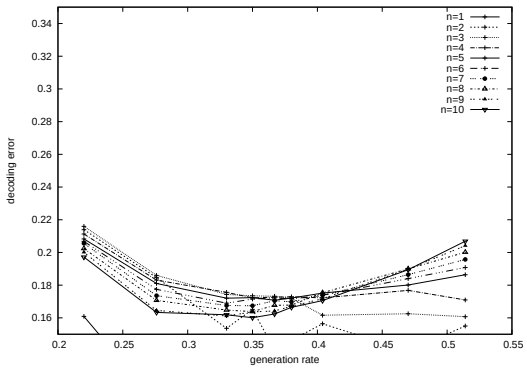

図 6　R=0..294 の時の復号誤り確率

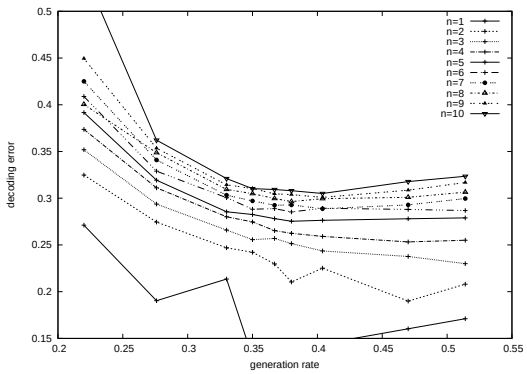

図 7　R=0.367 の時の復号誤り確率

4.2 検証結果

符号化レート R が通信路容量の 90% の R=0.330 の時のグラフを図 5 に，符号化レートが通信路容量の 80% の時のグラフを図 6 に，符号化レートが通信路容量と等しい時のグラフを図 7 に示す．

図 5 のグラフより符号語長 n が増えていくと，生成レートが 0.37 付近で復号誤り確率が最も小さくなることが確認できた．そして図 6，7 のグラフも図 5 のグラフと同様に生成レートが 0.37，0.38 付近で復号誤り確率が最も小さくなっていることが確認できる．

これらのグラフより符号化レート R の値を 3 つの値に固定し，生成レートを変化させて検証を行ったが，通信路容量より小さい符号化レート R の値では復号誤り確率が最も小さくなる生成レートは，符号化レート R の値に関わらず，式 (10) より導き出される生成レートの値と等しいことが検証できた．

5 まとめ

本研究では，パケットの時間間隔を利用し情報を送信することに着目した．そして，従来研究 [1] では示されていない出力レートと復号誤り確率の関係を検証するために，ランダム符号化により符号語を生成させ，従来研究 [2] によって最適な尤度関数と示された距離関数 3 を用いて，符号化レートを固定したうえで復号誤り確率を測定した．その結果，符号化レートの値に関わらず，最適な生成レートである 0.37 付近で復号誤り確率が最も小さくなることが分かった．

本研究では符号器，復号器を実際には設計せず，ランダム符号化を用いたシミュレーション実験で符号の性能を評価した．今後は実際の社会で実装されるためにコストや規模なども考慮したうえで符号器，復号器を作成したいと考えている．

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] 飯島 一貴, 「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」, 信州大学工学部, 学士論文, 2015.
[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　3 つの距離関数を用いて復号誤り確率を測定するプログラム

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <stdlib.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();
}

double ernd(double exp)
{
    double ret = -log(1 - rnd()) / exp;
    //printf("%f\n", ret);
    return(ret);
```

```
}

void
transmit(int n, double snd[], double rcv[])
{
    double curr = 0.0;
    double end = 0.0;
    int num = 0;
    int i_snd = 0;
    int i_rcv = 0;

    while (i_rcv < n) {
        if (num == 0 || (i_snd < n && snd[i_snd] < end)) {
            curr = snd[i_snd++];
            if (num == 0) {
                end = curr + ernd(1.0);
            }
            num++;
        }
        else {
            curr = end;
            num--;
            if (num > 0) {
                end = curr + ernd(1.0);
            }
            rcv[i_rcv++] = curr;
        }
    }
    return;
}


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


//距離関数 1
double
dist1(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;
        //printf("dist = %f\n", dist);
    }
    return(1.0 / dist);
}

//距離関数 2
double
dist2(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;
            //printf("%f\n", dist);
        }
    }
    return(1.0 / dist);
}

//距離関数 3

double
dist3(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] - my_max(rcv[i - 1], snd[i]);
                //printf("%f\n", dist);
            }
        }
    }
    if (dist == 0) {
        //printf("dist = 0\n");
        return(DBL_MAX);
    }
    return(1.0 / dist);
}

#define TRIAL 1000000
#define CODING_RATE 0.367
int
main(int argc, char *argv[])
{
    double EXPPARA;
    int n_from, n_to;
    double codeword[30];
    double rcv[30];
    double d_1, d_2, d_3;
    int n;
    int i;
    int j;
    int k_1, k_2, k_3;
    int trial;
    int dec_err1;
    int dec_err2;
    int dec_err3;
    int message_size;

    init_rnd(1234);


    if (argc != 4) {
```

```
    printf("困ります\n");
    return(0);
}


n_from = atoi(argv[1]);
if (n_from == 0) {
    printf("n_from = 0\n");
    return(0);
}

n_to = atoi(argv[2]);
if (n_to == 0) {
    printf("n_to = 0\n");
}

EXPPARA = atof(argv[3]);
if (EXPPARA == 0) {
    printf("EXPARRA = 0\n");
    return(0);
}

for (n = n_from; n <= n_to; n++) {

    dec_err1 = dec_err2 = dec_err3 = 0;

    message_size = exp(n * CODING_RATE / EXPPARA);

    for (trial = 0; trial < TRIAL; trial++) {

        codeword[0] = ernd(EXPPARA);
        for (i = 1; i < n; i++) {
            codeword[i] = codeword[i - 1] + ernd(EXPPARA);
        }

        transmit(n, codeword, rcv);

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

        //オリジナルの dist を測る
        d_1 = dist1(n, codeword, rcv);
        d_2 = dist2(n, codeword, rcv);
        d_3 = dist3(n, codeword, rcv);
        //printf("符号語との尤度: %f, %f, %f\n", d_1, d_2, d_3);

        k_1 = k_2 = k_3 = 0;
        for (i = 1; i < message_size; i++) {
            double d;

            //ダミー符号語作成
            codeword[0] = ernd(EXPPARA);
            for (j = 1; j < n; j++) {
                codeword[j] = codeword[j - 1] + ernd(EXPPARA);
                //printf("%f\n", codeword[j]);
            }

            if (k_1 == 0) {
                d = dist1(n, codeword, rcv);
                if (d_1 < d) {
                    dec_err1++;
                    k_1 = 1;
                }
            }
            if (k_2 == 0) {
                d = dist2(n, codeword, rcv);
                if (d_2 < d) {
                    dec_err2++;
```

```
                    k_2 = 1;
                }
            }
            if (k_3 == 0) {
                d = dist3(n, codeword, rcv);
                if (d_3 < d) {
                    dec_err3++;
                    k_3 = 1;
                }
            }
            if (k_1 == 1 && k_2 == 1 && k_3 == 1) {
                break;
            }
        }
    }
    //printf("dec_err1: %d(%f)\n", dec_err1, (double)dec_err1 / TRIAL);
    //printf("dec_err2: %d(%f)\n", dec_err2, (double)dec_err2 / TRIAL);
    //printf("dec_err3: %d(%f)\n", dec_err3, (double)dec_err3 / TRIAL);
    printf("%f,%f,%f\n", (double)dec_err1 / TRIAL, (double)dec_err2 / TRIAL, (double)dec_err3 / TRIAL);
}

return(0);

}
```