信州大学工学部

学士論文

実時間基準に基づく算術符号を用いた伝送システムにおける遅延の分析

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　06T2078C
氏名　藤山 直貴

2010 年 3 月 31 日

目次

1 **序章** 1

1.1 はじめに 1

1.2 研究の概要 1

2 **実時間基準について** 2

2.1 実時間基準とは 2

2.2 情報源と喪失確率 2

3 **システムにおける遅延** 3

3.1 実験用プログラム 3

3.2 入力シンボルの到着レート λ を変化させた際の遅延 4

3.3 符号化・復号化による遅延 4

3.4 送信バッファにおける遅延 6

3.5 送信バッファ内のシンボル数の分布 7

4 **まとめ** 9

謝辞 9

参考文献 9

付録 A 算術符号を用いた遅延を測るプログラム 11

付録 B 平均遅延を算出するプログラム 18

1　序章

1.1　はじめに

現在，インターネットなどの情報通信システムにおいて，伝送されるデータ量は莫大になってきている．これらの莫大なデータを，そのまま伝送することは不可能である．そこで，この莫大なデータを伝送する際に, 効率よくするにはどうすべきか，本研究ではデータを圧縮しかつ遅延を最小限にすることにより最適にデータを伝送することを考えた．符号化を行う中で，エンコーダに入力シンボルが到着しつつ符号化しながら，デコーダでエンコーダから符号語を受け取りつつ復元することができれば，よりリアルタイムなデータ伝送ができる．これを実現する符号化の一つに算術符号があり，これを用いることによって遅延も最小限に留めることができるのではないかと考えた [1]．さらに本研究ではエンコーダで符号化する際に，情報源におけるシンボルの出現確率から新たな符号化確率を算出し符号化することによって，システムにおける遅延にどのような影響を及ぼすかを考察した．また，本研究では，データがランダムな間隔で送られてくるというシステムが考察の対象であるが，このような不規則なデータ送信一般を簡単化して考えている．まずはじめに本研究の概要について述べる．

1.2　研究の概要

本研究で考えた算術符号のシステムのモデルを図 1 に示した．本研究において情報源は，情報源シンボル 0，1 をもち，レート λ のポアソン到着でシンボルを出力する．情報源から出力されたシンボルは算術符号によるエンコーダに入力される．そこでは，情報源シンボルの出現確率をそのまま使って符号語を生成するのでなく，出現確率から新たな符号化確率を計算し，それを用いて符号語を生成し圧縮する．これらの符号語は送信バッファに貯められ，そこから送信レート R でデコーダに送られる．すなわち 1 符号シンボルを送信するのに要する時間は $1/R$[秒] である．デコーダは符号語を受け取りながら情報源系列を復元していく．つまりエンコーダに全ての入力を待たずしてデコーダは復元を開始することができるのである．

本研究における遅延とは，一つのシンボルがエンコーダに入力されたときから，符号化され送信バッファに送られてまたここからデコーダに送られシンボルが復元されるまでに要する時間をいう．

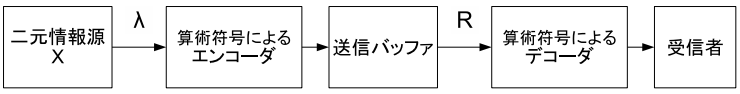

図 1　算術符号のシステム

従来の研究ではエンコーダにおいて，符号化確率として情報源シンボルの出現確率を用いていた（以下，これを情報源確率の場合と呼ぶ）．本研究では符号化確率として，次章で示す実時間基準における喪失確率を最小化する符号化確率を用いた（以下，これを実時間基準の場合と呼ぶ）．このように符号化確率を変えたことによってシステム全体の遅延にどのような影響を及ぼすか，それぞれの平均遅延を測定し考察した．

また, 先の実験で求めたシステム全体の遅延の内訳として，その要素の一つである算術符号の符号化・復号化よる遅延について調べ，符号化確率を変えたことによってどのような変化が起きたかを実験２で考察した．

システム全体の遅延とは先程の符号化・復号化による遅延と送信バッファにおける遅延の和であるため，この送信バッファにおける遅延についても実験３で測定し，より深く考察した．

当初の予想では，本研究で用いた符号化確率で符号化することによってバッファ内でのシンボル同士の衝突，即ち符号化されたシンボルがエンコーダからバッファに送られたときに，バッファ内にシンボルが存在しない確率が大きくなると予想していた．そこで, 符号化されたシンボルがエンコーダからバッファに送られたときにバッファ内に存在するシンボルの数の分布を実験４として調べた．その結果，到着レートが小さい場合を除いて予想通りではなかったため原因について考察した．

2　実時間基準について

2.1　実時間基準とは

情報の伝達における優先事項は，それぞれの状況に応じてさまざまである．それらすべてに共通する基準は伝送時の誤り確率を最小にすることであるが，それに加えて実時間通信においては，常に最新の情報を伝えたいという場合がある．気象情報の観測器を例に挙げて考えると，優先すべき情報は現地の過去の気象情報ではなく，現在どのような状況にあるかということである．それによって災害等を未然に防ぐことも可能になってくる．したがって，前の状態を誤りなく伝えることよりも現在の状態を伝えることの方が優先される．このような基準を実時間基準として定める [2]．観測器とその状態はそれぞれ情報源とそこから出力されるシンボルに対応している．

2.2　情報源と喪失確率

シンボルは情報源からレート λ のポアソン過程 $N(t)$ に従い出力される．シンボルの種類はポアソン過程とは独立な分布 P をもつ確率変数 X で表されるとする．さらに，このシンボルは有限集合 $\mathcal{X}$ から選ばれるものとする．また，通信路は単位時間あたりに長さ R の符号語を送る能力を持っているとする．

ここで，先ほど述べた実時間基準に従えば，時刻 t に出力されるシンボル X に対する符号語を送信中に次のシンボルが出力されたときは，現在の符号語の送信を中止し，新しいシンボルに対する符号語を優先的に送信する．このときひとつ前のシンボル X は受信者に正しく伝わらない．この事象が起こる確率を実時間基準における喪失確率とする．この喪失確率とは本研究において，エンコーダで符号化されたシンボルがバッファに送られる際にバッファ内に一つ以上のシンボルが存在する確率に等しいと考えられる．この喪失確率を最小化することを考えると，[2] により，これを達成する符号化確率 $P'(x)$ は

$$P'(x) = \frac{P(x)^{\frac{R}{R-\lambda}}}{\sum_{\tilde{x}\in\mathcal{X}} P(\tilde{x})^{\frac{R}{R-\lambda}}} \tag{1}$$

と求められている．この符号化確率を用いることで，実時間基準における喪失確率を最小化する，すなわち，バッファ内でのシンボルの衝突の確率を最小化することが予想される．また，実時間基準のモデルではシンボルの喪失確率を最小化することについて考察しているため，それぞれのシンボルの遅延については考察されていない．実時間基準のモデルでは，バッファにシンボルが溜め込まれるという現象を考えておらず，そういった場合は送信中のシンボルを喪失させて新しく符号化されたシンボルを送信するようになっている．しかし本研究のモデルではそれぞれのシンボルは一度送信バッファの中に溜め込まれ，先に溜め込まれたシンボルから順に送信されるため，送信される全てのシンボルを遅延の対象としている．この両者の違いが本研究の結果に大きく影響していると考えられる．また入力シンボルの到着レート λ とバッファの送信レート R を同時に大きくすることは，システム全体の動きを早くするということと同等である [1] ため，本研究では $R = 1$ と固定した上で実装し，従来の研究と比較・考察を行った．

3 システムにおける遅延

3.1 実験用プログラム

本研究における情報源からの入力シンボルの数は 100,000 個である．2 つのシンボル 0，1 の出現確率をそれぞれ $p, 1-p$ とするために Knuth の乱数発生法 [3] で $[0, 1)$ 上の一様乱数 u を発生させ，$u < p$ ならば 0，$u \geq p$ ならば 1 を発生させた．また，パラメータ λ の指数分布は密度関数 $f(x) = \lambda e^{-\lambda x}$ に従う．よって分布関数は $F(x) = \int_0^x f(t)dt = 1 - e^{-\lambda x}$ となる．これより，逆関数 $F^{-1}(u) = -\frac{1}{\lambda}\log_e(1-u)$ の u に [0,1) 上の一様乱数を代入することで指数分布を得る．一様乱数は Knuth の乱数発生法で発生させる．

遅延については，各情報源シンボルがエンコーダに入力された時刻を記録しておき，デコーダにより復元された時刻との差をとることにより測定している．

3.2 入力シンボルの到着レート λ を変化させた際の遅延

従来の研究では情報源のシンボルの出現確率をそのままエンコーダの符号化確率としていたが，本研究では実時間基準における喪失確率を最小化する符号化確率を式 (1) から算出し，これを用いて両者の平均遅延を比較した．

3.2.1 実験内容

情報源のシンボルにおいて，0 の出現確率 $P(0)$ を 0.1，0.2，0.3，0.4 とし，シンボルの到着レート λ を変えながら実時間基準の場合と情報源確率の場合，それぞれの平均遅延を測定した．この際，到着レート λ の範囲は実時間基準の場合と情報源確率の場合と同時に測定するため，実時間基準の場合において送信バッファがパンクした値までとして測定した．

3.2.2 結果と考察

$P(0)$=0.1，0.2，0.3，0.4 としたとき，到着レート λ に対するシステム全体の平均遅延をそれぞれ図 2，図 3，図 4，図 5 に示した．尚，グラフ中の new は実時間基準の場合，old は情報源確率の場合をそれぞれ示している．図から分かるように実時間基準の場合と情報源確率の場合，両者の平均遅延の値には大きな差は生じなかった．詳しく値を見てみると若干ではあるが実時間基準の場合の平均遅延の方が大きな値となっている．また，実時間基準の場合の平均遅延の傾向として，到着レート λ が 0.7 付近になると急激に大きくなっていることが分かる．これは，符号化確率を変えたことによって符号語長のエントロピーが大きくなったため，どんどんバッファ内にシンボルが溜まっていき，情報源確率の場合より小さな λ の値で送信バッファがパンクという現象を起こしてしまうからである．この現象については後に述する．

3.3 符号化・復号化による遅延

先程の実験でもとめたシステム全体の遅延の要素の一つとして，算術符号の符号化・復号化によるものがある．そこで，これについても実時間基準の場合と情報源確率の場合，それぞれの遅延を測定し，結果にどのような影響を及ぼしているかを確認した。

3.3.1 実験内容

平均遅延を求める実験において，送信レート $R = 1$ と固定していたが，送信レート $R = \infty$ とすることで，バッファにおける遅延が生じないようにし，算術符号の符号化・復号化による

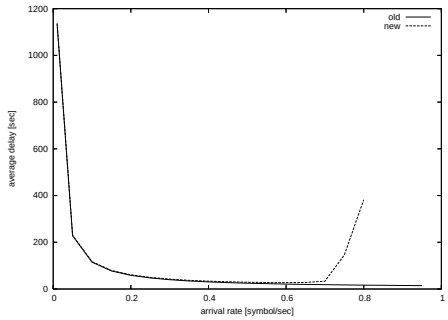

図 2 平均遅延 P(0)=0.1

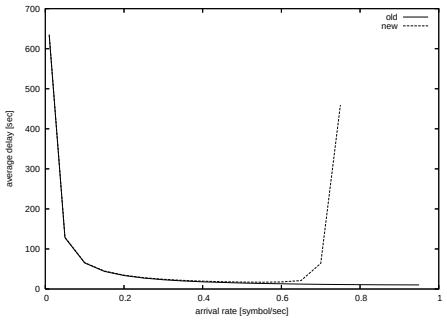

図 3 平均遅延 P(0)=0.2

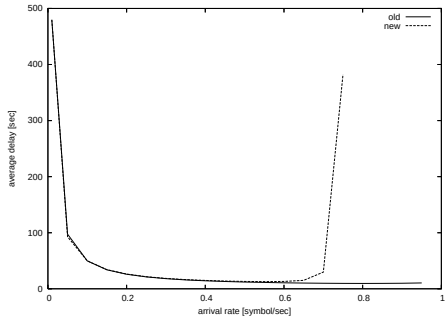

図 4 平均遅延 P(0)=0.3

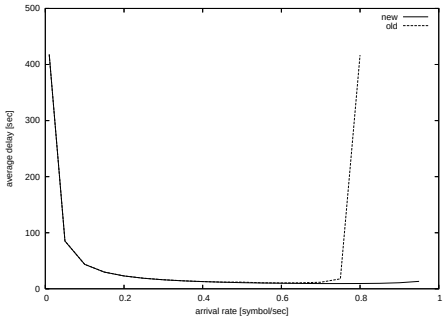

図 5 平均遅延 P(0)=0.4

遅延を測定した.

3.3.2 結果と考察

$P(0)$=0.1，0.2，0.3，0.4 としたときの符号化・復号化による遅延をそれぞれ図 6，図 7，図 8，図 9 に示した．尚，グラフ中の new は実時間基準の場合，old は情報源確率の場合をそれぞれ示している.

図から分かるように算術符号の符号化・復号化による遅延についても実時間基準の場合と情報源確率の場合，両者の平均遅延の値には大きな差は生じなかった．また，これについても詳しく値を見てみると若干ではあるが実時間基準の場合の遅延の方が大きな値となっていた．これは，符号化確率を変えたことによって符号語長が長くなり，算術符号において区間の仕分けに時間がかかってしまったためであると考えられる.

3.4 送信バッファにおける遅延

システム全体の遅延の要素の一つとして，算術符号の符号化・復号化によるものがあったが，そのほかにバッファにおける遅延がある．エンコーダで符号化されたシンボルがバッファに送られると送信レート R でデコーダに送られるが，この送信レートによりシンボルがバッファに溜め込まれる現象が起きてしまう．そこで，このバッファにおける遅延についても測定し，符号化確率を変えたことで結果にどのような影響を及ぼしているかを調べた．

3.4.1 実験内容

システム全体の遅延の構成要素は符号化・復号化による遅延とバッファにおける遅延の和であるので，これまでの実験で求めたそれぞれの結果から，バッファにおける遅延を算出した．

3.4.2 結果と考察

$P(0)$=0.1，0.2，0.3，0.4 としたときのバッファにおける遅延をそれぞれ図 10，図 11，図 12，図 13 に示した．尚，グラフ中の new は実時間基準の場合，old は情報源確率の場合をそ

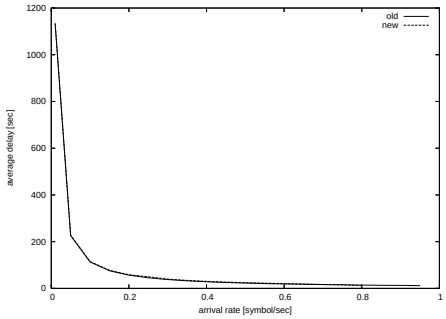

図 6 符号化・復号化による遅延 P(0)=0.1

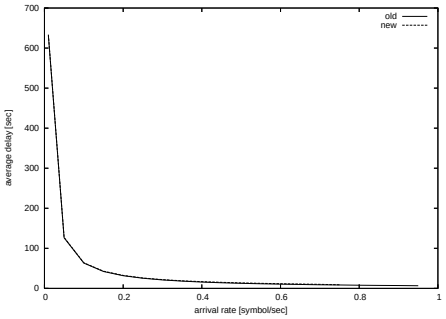

図 7 符号化・復号化による遅延 P(0)=0.2

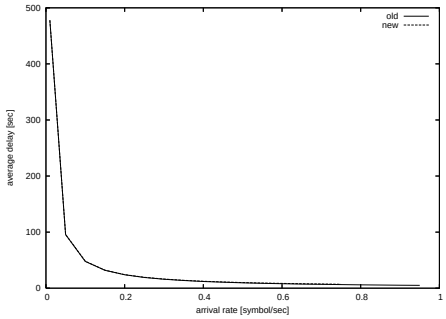

図 8 符号化・復号化による遅延 P(0)=0.3

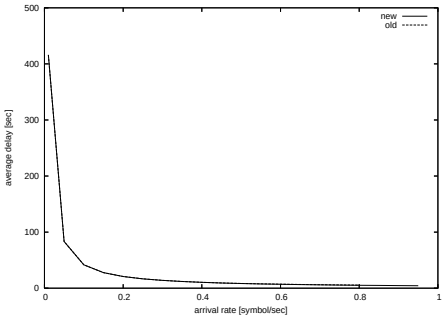

図 9 符号化・復号化による遅延 P(0)=0.4

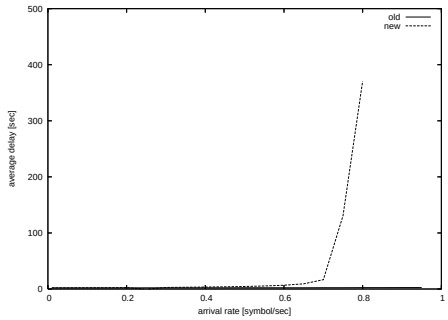

図 10 バッファにおける遅延 P(0)=0.1

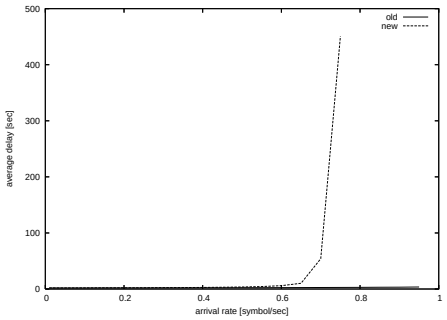

図 11 バッファにおける遅延 P(0)=0.2

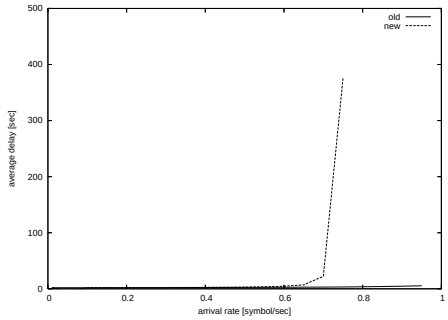

図 12 バッファにおける遅延 P(0)=0.3

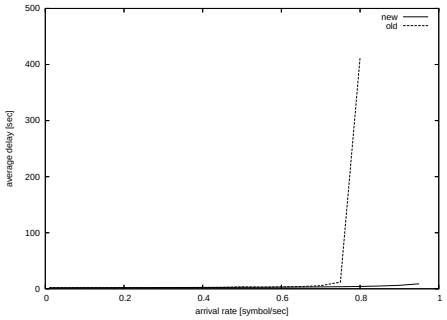

図 13 バッファにおける遅延 P(0)=0.4

れぞれ示している．図から分かるように，情報源確率の場合は到着レート λ が 1 以下ではバッファはパンクしない．それに比べて，符号化確率を変えると，それに伴い符号語長のエントロピーが大きくなってしまうので λ が 0.8 付近になるとバッファがパンクしてしまう．このパンクという現象はバッファ自身が持つ送信能力よりバッファに溜め込まれる符号語の方が大きくなってしまう為，バッファの許容量を越えてしまうことをさす．従来のものよりも到着レートが小さい値でパンクしてしまう為，システムとしても改良されたとは決していえない．

3.5 送信バッファ内のシンボル数の分布

先程の実験で求めたバッファにおける遅延についてさらに詳しく調べる為に，エンコーダで符号化されたシンボルがバッファに送られたときに，バッファ内に未送信のシンボルがいくつあるか，その分布を調べた。

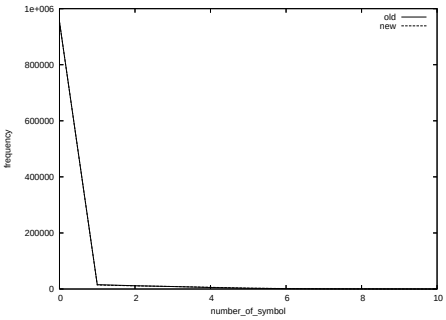

図 14 バッファ内のシンボル数分布 λ=0.1

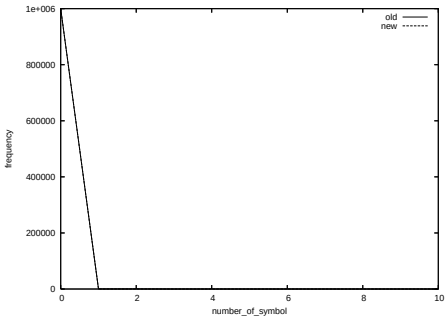

図 15 バッファ内のシンボル数分布 λ=0.001

3.5.1 実験内容

エンコーダで符号化されたシンボルがバッファに送られたときに，バッファの中に溜め込まれているシンボルの数を調べ，それがどのくらいの頻度で起こるかを調べた．この際，到着レート λ は 0.1 と 0.001 の 2 通りで確認した．

3.5.2 結果と考察

λ=0.1，0.001 としたときのバッファ内に溜め込まれているシンボルの数の分布をそれぞれ図 14，図 15 に示した．尚，グラフ中の new は実時間基準の場合，old は情報源確率の場合をそれぞれ示している．

この結果からも，両者に大きな差は生じなかった．当初の予想では，今回用いた符号化確率で符号化することによってバッファ内でのシンボル同士の衝突，即ちエンコーダから符号化されたシンボルがバッファに送られたときに，バッファ内にシンボルが存在しない確率が小さくなると予測していた．この予測と違う結果となった原因として，次のことが考えられる．今回用いた符号化確率は実時間基準における喪失確率を最小化するものであった．この実時間基準における喪失というのは，現在送信中のシンボルと次に送られるシンボルの 2 シンボルの間での考察である．しかし，本研究に用いたモデルではバッファがあるため常に 1 つのシンボルが影響を受けるのは他の 1 つのシンボルだけではない．仮に 2 つのシンボルの間では衝突が起きない可能性があったとしても，先に送信されているはずのシンボルがそのさらに先のシンボルとの衝突によって，まだ送信されずにバッファ内に溜め込まれてしまう．これによって，シンボル同士の衝突が多く起こってしまう．さらに今回用いた符号化確率によって符号語長のエントロピーが大きくなってしまっているため，この現象が起きてしまう確率も増えてしまっている．この結果が顕著に現れているのが図 14 であり，送信レート $R = 1$ と固定であるのに対

し到着レート λ の比が大きい図 14 では符号化確率をかえたことによって，バッファ内に溜め込まれているシンボルの数が 0 である頻度が小さくなっている．逆に，図 15 では図 14 に比べてさらに到着レート λ の値を小さくすることによって，バッファ内にシンボルが溜まりにくくしている．これによって，2 シンボルの間での考察がしやすい状況となっているため，情報源確率の場合に比べてバッファ内に溜め込まれているシンボルの数が 0 である頻度が大きくなっている．

4　まとめ

本研究では，算術符号によって符号化・復号化するシステムの中で，情報源のシンボルの出現確率をそのまま符号化確率としていた従来の研究を参考に，実時間基準という新たな考察の元で導き出した符号化確率を使うことにより，システム全体にどのような影響を及ぼすか検証を行った．当初の予想では，実時間基準における喪失というのが，本研究におけるモデルにおいてバッファ内でのシンボルの衝突を表していて，喪失確率を最小化する符号化確率を用いることによって，バッファにおける遅延が小さくなるのではないかと考えていた．しかし，システム全体の遅延にしろ，符号化・復号化による遅延，バッファにおける遅延についても改善されなかった．さらには符号化確率を変えたことによって，符号語長のエントロピーが大きくなり，情報源確率の場合より λ が小さい値でバッファがパンクという現象を起こしてしまった．これらの原因となったのが，考察する際の条件の違いであったと考えられる．実時間基準においては，送信中のシンボルと次にくるシンボルの 2 つのシンボルの間だけで考察していたが，本研究で用いたモデルのように実際にはバッファというものが存在する．このバッファの中にシンボルが 1 つも無い，もしくは 1 つしかない場合は，実時間基準と同様に 2 つのシンボルの間だけで考察できるが，2 つ以上のシンボルが存在した場合，それは既に実時間基準と同様には考えられない．

本研究で分かったことは，より良いシステムの設計をする際，実際に実験に用いるモデルと考察するモデルの条件が一緒であるかを確認することが重要だということだ．本研究のようにその条件に違いがあると，思ったような結果が出ないときがある．

謝辞

本研究を行うにあたり，多大な支持と指導をしてくださった西新幹彦准教授と西新研究室の皆様に感謝の意を表する．

参考文献

[1] 杉原辰徳, “算術符号を用いた伝送システムにおけるポアソン到着シンボルの遅延”, 信州大学工学部卒業論文, 2008.

[2] 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”, SITA2003 予稿集, pp153-154, 2003.

[3] 奥村晴彦, C 言語による最新アルゴリズム辞典, 技術評論社, 1991.

付録 A　算術符号を用いた遅延を測るプログラム

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

#define CODE_VALUE_BITS 16
#define TOP_VALUE (((long)1 << CODE_VALUE_BITS) - 1)    /* 符号化の精度 */

#define FIRST_QTR (TOP_VALUE / 4 + 1)
#define HALF (2 * FIRST_QTR)
#define THIRD_QTR (3 * FIRST_QTR)

#define NO_OF_CHARS (2)                                 /* シンボルの種類 */
#define MAX_FREQUENCY (16383)

double lambda;                                          /* 到着レート λ */


/*********************************************************************************/

/* 指数分布乱数を発生 */

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

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 rand_(void)
{
        return (-log(1-(rnd()))/lambda);                /* λの指数関数 */
}


/*****************************************************************************/


double present_time;                                    /*   現在時刻   */
double arrival_time;                                    /*   到着時刻   */
double sending_time;                                    /* 送信終了時刻 */

#define ARRIVING_LIST_MAX 1000
#define SENDING_BUFFER_LIST_MAX 1000
int next_ch;

double arriving_list[ARRIVING_LIST_MAX];                /* 到着関係 */
int arriving_count=0;
int arriving_in=0;
int arriving_out=0;

int sending_buffer_list[SENDING_BUFFER_LIST_MAX];       /* 送信関係 */
int sending_buffer_count=0;
int sending_buffer_in=0;
int sending_buffer_out=0;

double prob0;                                           /* シンボル 0 の出現確率 */
double cum_freq[NO_OF_CHARS + 1];                       /* 累積出現確率 */

#define NUMBER_OF_SYMBOLS 1000000                       /* 入力シンボルの数 */


/*****************************************************************************/


void add_arriving_list(double time)                     /* 到着リスト処理 */
{
        if (arriving_count >= ARRIVING_LIST_MAX)
        {
                printf("warning1!!");
                exit(0);
        }
        arriving_list[arriving_in] = time;
        arriving_in = (arriving_in + 1) % ARRIVING_LIST_MAX;
        arriving_count++;
        return;
}

double get_arriving_list()
{
        double time;

        if (arriving_count == 0)
        {
                printf("warning2!!");
                exit(0);
        }
        time = arriving_list[arriving_out];
        arriving_out = (arriving_out + 1) % ARRIVING_LIST_MAX;
        arriving_count--;
```

```
        return(time);
}

void add_sending_buffer_list(int bit)
{

        if (sending_buffer_count >= SENDING_BUFFER_LIST_MAX)
        {
                printf("warning3!!");
                exit(0);
        }
        sending_buffer_list[sending_buffer_in] = bit;
        sending_buffer_in = (sending_buffer_in + 1) % SENDING_BUFFER_LIST_MAX;
        sending_buffer_count++;
        return;
}

int get_sending_buffer_list()
{
        int bit;

        if(sending_buffer_count == 0)
        {
                printf("warning4!!");
                exit(0);
        }
        bit=sending_buffer_list[sending_buffer_out];
        sending_buffer_out=(sending_buffer_out+1)%SENDING_BUFFER_LIST_MAX;
        sending_buffer_count--;
        return(bit);
}


/*****************************************************************************/


void
start_model()                                           /* 累積出現確率の計算 */
{
        cum_freq[0] = 0;
        cum_freq[1] = pow(prob0, 1.0/(1.0-lambda));
        cum_freq[2] = pow(prob0, 1.0/(1.0-lambda)) + pow((1.0-prob0), 1.0/(1.0-lambda));

/*      if (cum_freq[NO_OF_CHARS] > MAX_FREQUENCY)
                abort();
*/
        return;
}


/*****************************************************************************/


int low;
int high;
long bits_to_follow;
unsigned long input_length=0;
unsigned long output_length=0;

void
start_encoding()                                        /* 初期設定（エンコーダ）*/
{
        low = 0;
        high = TOP_VALUE;
        bits_to_follow = 0;
        return;
}
```

```
/*****************************************************************************/


int d_low;
int d_high;
int v_low;
int v_high;

void
start_decoding()                                        /* 初期設定（デコーダ）*/
{
         d_low=0;
         d_high=TOP_VALUE;
         v_low=0;
         v_high=TOP_VALUE;
         return;
}


/*****************************************************************************/


void
decode_symbol(int bit, double cum_freq[])               /* 算術符号（復号化）*/

{
        if( bit==0 )
        {
                v_high=(v_high + v_low+1)/2-1;
        }
        else
        {
                v_low=(v_high + v_low+1)/2;
        }
        for(;;)
        {
                int d_b;
                double delay;

                d_b=(long)((d_high-d_low+1)*(cum_freq[1]/cum_freq[2]))+d_low;

                /* 同じ数どうしで割らないと値がずれることがある */

                if (v_high+1 <= d_b)
                {
                        delay = present_time - get_arriving_list();

                        printf(" %f \n",delay);                 /* 遅延 */

                        output_length++;
                        d_high=d_b-1;
                }
                else if (v_low>=d_b)
                {
                        delay = present_time - get_arriving_list();

                        printf(" %f \n",delay);                 /* 遅延 */

                        output_length++;
                        d_low=d_b;
                }
                else
                {
                        break;
                }
                for (;;)
                {
                        if (d_high+1<=HALF)
                        {
```

```
                                d_low=d_low*2;
                                d_high=(d_high+1)*2-1;
                                v_low=v_low*2;
                                v_high=(v_high+1)*2-1;
                        }
                        else if (d_low>=HALF)
                        {
                                d_low=(d_low-HALF)*2;
                                d_high=(d_high-HALF+1)*2-1;
                                v_low=(v_low-HALF)*2;
                                v_high=(v_high-HALF+1)*2-1;
                        }
                        else if(d_low>=FIRST_QTR && d_high+1<=THIRD_QTR)
                        {
                                d_low=(d_low-FIRST_QTR)*2;
                                d_high=(d_high-FIRST_QTR+1)*2-1;
                                v_low=(v_low-FIRST_QTR)*2;
                                v_high=(v_high-FIRST_QTR+1)*2-1;
                        }
                        else
                        {
                                break;
                        }
                }
        }
        return;
}

void
sending_finish()                                        /* 送信終了処理 */
{
        int bit;

        bit = get_sending_buffer_list();
        decode_symbol(bit, cum_freq);                   /* デコード処理 */

        if(sending_buffer_count == 0)
        {
                sending_time = -1.0;
        }
        else
        {
                sending_time = present_time + 1.0 ;
        }
        return;
}

void
bit_plus_follow(int bit)
{
        add_sending_buffer_list(bit);

        if ( sending_time < 0.0)
        {
                sending_time = present_time + 1.0 ;
        }
        while (bits_to_follow > 0)
        {
                add_sending_buffer_list(1 - bit);
                bits_to_follow--;
        }
        return;
}

void
encode_symbol(int symbol,double cum_freq[])             /* 算術符号（符号化）*/
{
        long range;
```

```
        range = (long)(high - low) + 1;

        high = low + (long)(range * (cum_freq[symbol+1]/cum_freq[NO_OF_CHARS])) - 1;
        low  = low + (long)(range * (cum_freq[symbol  ]/cum_freq[NO_OF_CHARS]));

        /* 同じ数どうしで割らないと値がずれることがある */

        for(;;){

                if (high < HALF)
                {
                        bit_plus_follow(0);
                }
                else if (low >= HALF)
                {
                        bit_plus_follow(1);
                        low  -= HALF;
                        high -= HALF;
                }
                else if (low >= FIRST_QTR && high < THIRD_QTR)
                {
                        bits_to_follow += 1;
                        low  -= FIRST_QTR;
                        high -= FIRST_QTR;
                }
                else
                {
                        break;
                }

                low  = 2 * low;
                high = 2 * high + 1;
        }

        return;
}


/*****************************************************************************/


void
done_encoding()
{
        bits_to_follow += 1;
        bit_plus_follow((low < FIRST_QTR)? 0: 1);
        return;
}


/*****************************************************************************/


void
arrival_symbol()                                        /* 到着シンボル処理 */
{

        input_length++;

        add_arriving_list(present_time);

        encode_symbol(next_ch, cum_freq);               /* エンコード処理 */

        if (input_length >= NUMBER_OF_SYMBOLS)
        {
                arrival_time = -1.0;                    /* 負だと main のループ終了 */
        }
        else
        {
```

```
                arrival_time = present_time + rand_();
        }
        return;
}



int
main(int argc, char *argv[])
{

        if (argc != 3){
                printf("引数の数が違います \n");
                exit(1);
        }

        lambda = atof(argv[1]);
        prob0 = atof(argv[2]);

        init_rnd(1);

        present_time = 0.0;             /*   現在時刻   */
        arrival_time = rand_();         /*   到着時刻   */
        sending_time = -1.0;            /* 送信終了時刻 */



        start_model();
        start_encoding();
        start_decoding();


        while ((arrival_time >= 0.0) || (sending_time >= 0.0))
        {
                if (arrival_time<0 || (sending_time>=0 && arrival_time>sending_time))
                {
                        present_time = sending_time;
                        sending_finish();               /* 送信終了処理 */
                }
                else
                {
                        if(rnd() <= prob0)              /* 出現確率より */
                        {
                                next_ch = 0;            /* 到着シンボル 0 */
                        }else{
                                next_ch = 1;            /* 到着シンボル 1 */
                        }

                        present_time = arrival_time;
                        arrival_symbol();               /* 到着シンボル処理 */
                }
        }

        return(0);

}
```

付録 B　平均遅延を算出するプログラム

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define N_SLOTS (1 << 16)

double max_delay;
double min_delay;
long count[N_SLOTS];
double lower, upper, range;

double
psum(int i, int size)
{
        if (size == 1)
        {
                return((lower + (i + 0.5L) * range / N_SLOTS) * count[i]);
        }
        return(psum(i, size / 2) + psum(i + size / 2, size / 2));
}

int main()
{
        char buffer[20];
        double x;
        long number;
        double sum, expect;
        int i;
        FILE *fp;
        max_delay = 0;
        min_delay = 10000;

        if ((fp = fopen("data.txt", "rt")) == NULL)
        {
                printf("file open error!!\n");
                exit(EXIT_FAILURE);
        }

        while(fgets(buffer,sizeof(buffer),fp) != NULL)
        {
                x = atof(buffer);

                if(x > max_delay)
                {
                        max_delay = x;
                }
                if(x < min_delay)
                {
                        min_delay = x;
                }
        }
        fseek(fp,0,SEEK_SET);

        lower = floor(min_delay);
        upper = ceil(max_delay);
        range = upper - lower;

        for (i = 0; i < N_SLOTS; i++)
        {
                count[i] = 0;
        }
        number = 0;
```

```
        while(fgets(buffer,sizeof(buffer),fp) != NULL)
        {
                x = atof(buffer);
                number++;
                i = (int)(((x - lower) / range) * N_SLOTS);
                if (i >= 0 && i < N_SLOTS)
                {
                        count[i]++;
                }
                else
                {
                        printf("range error (%d)\n", number);
                        exit(1);
                }
        }
        fclose(fp);

        sum = psum(0, N_SLOTS);
        expect = sum / number;
        printf("%g\n", expect);

        return(0);

}
```