信州大学工学部

学士論文

ポアソン通信路に対する通信路容量の離散化近似計算に向けて

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　21T2121D
氏名　平松 里彩

2025 年 02 月 10 日

目次

1　はじめに

現代社会では，通信は情報のやり取りやコミュニケーションの基盤として不可欠な存在となっている．この通信において，情報は通信路を介して送受信されるが，その過程で送信された情報には雑音が加わり，受診者の元に届く．すなわち，送信者から送信された情報を誤りなく伝達することは不可能である．通信によって生じた誤りを検出，訂正する能力を高めると，一度の通信で伝達できる情報量は少なくなる．限りなく小さい復号誤り確率のもとで伝達できる情報量の最大値は「通信路容量」と呼ばれている．

通信には様々な方法がある．その中で，光子を放出し，その個数を基に通信を行うものがある．これを，光子計数通信システムと呼び，この光子計数通信システムの確率モデルはポアソン分布で表すことができる．本論文では，このような通信路をポアソン通信路と呼ぶ．

ポアソン通信路の通信路容量を求めるという問題は未解決であるが，特定の条件や制約のもとでの近似的な解析や数値的な計算が行われている [1]．先行研究では，ポアソン通信路の入力が指数分布に従うと仮定して通信路容量の下界を求め，ポアソン通信路で最低どれだけの情報量を送ることができるのかを明らかにしている [1]．本研究では，このポアソン通信路の通信路容量を Blahut アルゴリズムを用いて算出し，先行研究で算出された値との比較を行う．また，そのときの入力分布についても調査し，最適な入力分布を明らかにする．

2　ポアソン通信路の通信路容量

2.1　定常無記憶通信路の通信路容量

本節では，通信路容量について先行研究に基づき定義する．確率変数 X に対し，その確率分布を P_X と表す．そして，$X = x$ が与えられたもとで $Y = y$ となる条件付き確率を $P_{Y|X}(y|x)$，同時確率分布を $P_{XY}(x, y)$ と表す．数学的に，通信路は通信路入力 X が与えられると通信路出力 Y が確率的に決まるものとして条件付き確率 $P_{Y|X}(y|x)$ で表す．

X と Y の間の相互情報量 $I(X;Y)$ を次のように定義する．

定義 1

$$I(X;Y) \triangleq \sum_x \sum_y P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \tag{1}$$

定理 1 定常無記憶通信路の通信路容量は

$$C(P_{Y|X}) = \max_{P_X} I(X;Y) \tag{2}$$

で与えられる．

2.2 光子計数通信システム

次に，本研究で扱う光子計数通信システムについて述べ，それをポアソン通信路という確率モデルで定義する．

光子計数通信システムとは，光子を放出させ，その光子の個数を検出することで情報の伝達を行う通信システムのことである．通信の送信側で電力が与えられることにより光源からランダムに光子が放出される．与える電力を通信路入力 x，それによって放出される光子の個数を通信路出力 y として考えると，確率モデルは

$$P_{Y|X}(y|x) = \frac{1}{y!}x^y e^{-x}, \quad x > 0,\ y \in \{0, 1, \cdots\} \tag{3}$$

と書くことができる [1]．通信路の確率モデルがポアソン分布で表現されるものをポアソン通信路と呼ぶ．このように，光子計数通信システムはポアソン通信路とよばれる確率モデルで表現することができる．

3 コスト制約付き通信路容量の数値計算

本研究ではポアソン通信路の通信路容量を Blahut アルゴリズムを用いて近似計算した．本章では，文献 [2] に基づき Blahut アルゴリズムについて述べる．このアルゴリズムは，離散無記憶通信路の通信路容量を求めるためのもので，入力確率分布を更新することで相互情報量を最大化し，その結果として通信路容量が求められる．アルゴリズムは以下のように進行する．

1. パラメータ s の値を定める．s は，0 以上の値を取る．
2. 入力確率分布 $P_X(x)$ を一様分布で初期化する．
3. 出力確率分布 $P_Y(y)$ を

$$P_Y(y) = \sum_x P_X(x) P_{Y|X}(y|x) \tag{4}$$

で計算する．ここで，$P_{Y|X}(y|x)$ は通信路の遷移確率であり，本研究ではポアソン通信路の遷移確率となるため，式 (3) を用いる．

4. 入力確率分布を

$$P'_X(x) = \frac{P_X(x) \cdot \exp\left(\sum_y P_{Y|X}(y|x) \log\left(\frac{P_{Y|X}(y|x)}{P_Y(y)}\right) - s \cdot \mathrm{cost}(x)\right)}{\sum_x P_X(x) \cdot \exp\left(\sum_y \left(P_{Y|X}(y|x) \log \frac{P_{Y|X}(y|x)}{P_Y(y)}\right) - s \cdot \mathrm{cost}(x)\right)} \tag{5}$$

で更新する．ここで，$\mathrm{cost}(x)$ は入力アルファベットに対応するコスト関数であり，本研究では $\mathrm{cost}(x) = x$ とする．

5. 相互情報量 I_L と I_U の差が，指定した収束条件以下になるまで，3 と 4 を繰り返す．ここで，相互情報量 I_L が下限，I_U が上限であり，それぞれ

$$I_L = \log\left(\sum_x P_X(x)\cdot c(x)\right) \tag{6}$$

$$I_U = \log\left(\max_x c(x)\right) \tag{7}$$

で求められる．ここで，$c(x)$ は通信路の遷移確率や出力確率分布に基づく係数で，

$$c(x) \triangleq \exp\left(\sum_y P_{Y|X}(y|x)\log\left(\frac{P_{Y|X}(y|x)}{P_Y(y)}\right)\right) \tag{8}$$

と定義される．

6. 最終的な入力確率分布 P_X と出力確率分布 P_Y が定まる．また，

$$E = \sum_x P_X(x)\cdot \mathrm{cost}(x) \tag{9}$$

$$C = s\cdot\sum_x P_X(x)\cdot \mathrm{cost}(x) + I_L \tag{10}$$

により，コスト E と通信路容量 C が得られる．

4 入力アルファベットの離散化

前章では，Blahut アルゴリズムについて説明した．しかし，Blahut アルゴリズムは離散無記憶通信路，すなわち離散的な入力アルファベットを持つ通信路の通信路容量を求めるためのものであるのに対して，ポアソン通信路の入力アルファベットは正の実数であり，したがって連続である．このままでは Blahut アルゴリズムを用いた計算を行うことができないため，入力アルファベットを離散化した上で近似計算する必要がある．本研究で採用した離散化の方法を説明する．

まず，正数全体を j 個に離散化することを考える．パラメータ λ を持つ指数分布

$$f(x) = \lambda e^{-\lambda x}, \quad x > 0 \tag{11}$$

より，分布関数

$$F(x) = \int_0^x f(x)dt = 1 - e^{-\lambda x} \tag{12}$$

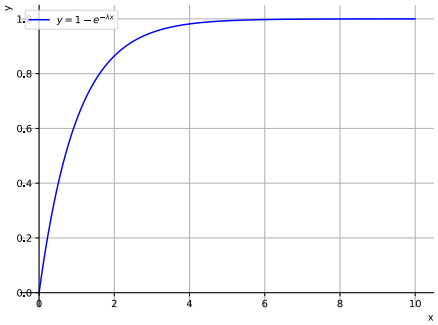

図 1　$\lambda = 1$ の分布関数

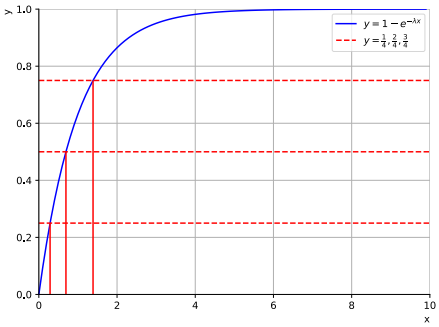

図 2　3 個に離散化した場合

が得られる．y 軸を $j+1$ 等分し，$F(x)$ との交点を j 個取ることができる．このときの x 座標を採用し，離散化が完了する．得られる離散化後の x は，

$$x = \frac{1}{\lambda}(\log(j+1) - \log(j+1-z)) \tag{13}$$

で与えられる．ここで，z は離散化後の x の番号であり，0 から j の範囲の整数値をとる．λ を 1 とした場合の分布関数を図 1 に，j を 3 とした場合を図 2 に示す．y 軸を 4 等分し，$F_x(x)$ との交点を 3 点取ることができることが分かる．

ポアソン通信路において，コストが大きくなるほど通信路容量が大きくなるのは明らかである．$\mathrm{cost}(x) = x$ より，入力アルファベット x が大きくなるほどコスト E は大きくなる．本研究では通信路容量を近似するため，コスト E の値，すなわち x の値は小さい方が望ましい．この離散化方法において，指数分布を用いることで，λ の値を大きくするほど，離散後の x の値を小さくすることができる．しかし，x の値が小さくなると，式 (3) の x^y の値は非常に小さくなり，計算の際に不具合が生じた．そのため，本研究では，不具合が確認されなかった $\lambda = 1$ を採用した．

5　検証結果と考察

5.1　コストあたり通信路容量の算出

本研究では，入力は正数全体から 40 点を選び，出力は非負整数全体から $0, 1, \cdots, 39$ と 40 点を選んだ．そして，3 章で説明したパラメータ s の値を変化させてコストあたりの通信路容量を導出した．入力の値は，選ぶ数を増やすほど各 x の値同士の間隔が狭くなり，式 (4)(5) の確率の更新量が微小になる．そのため，3 章の手順 5 で説明した収束条件を満たすことなく定めた反復回数 10,000 回を超えてしまう．このこととプログラムの実行時間を考慮し，40 という値を決定した．

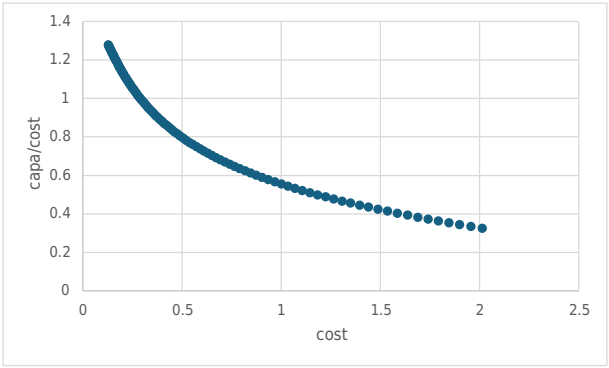

図 3　パラメータ $0.0 \leq s < 1.0$ のときのコストあたり通信路容量

s を 0 から 1.0 の間を 0.01 ずつ変化するとした場合のコストあたり通信路容量をグラフで表したものを図 3 に示す．グラフは単調減少となっており，コストあたり通信路容量は，コスト 0 付近で最大値を取ると考えられる．パラメータ s を大きくするほどコストの値が小さくなっており，このときのコストあたり通信路容量の最大値は $s = 0.99$ のときの 1.279 ビット/エネルギーであった．

よりコスト 0 付近の近似的な計算を行うために，他の条件はそのままでパラメータ s の値を大きくし，再度計算を行った．s を 1.0 から 2.0 の間を 0.01 ずつ変化するとした場合のコストあたり通信路容量をグラフで表したものを図 4 に示す．図 3 でみられた単調減少の特徴がみられなくなっている．このときのコストあたり通信路容量の最大値は $s = 1.41$ のときの 1.410 ビット/エネルギーであった．パラメータ s の値を大きくし，よりコスト 0 に近づけた場合の値を求めることも試みたが，3 章の手順 4 で $P'_X(x)$ を更新する際，式 (5) の分子，分母にある exp 内の式の値が収束し，更新量が微小になる．そのため，3 章の手順 5 で説明した収束条件を満たさず，指定した反復回数 10,000 回以内で計算を行うことが不可能であった．よって，本論文では 1.410 ビット/エネルギーを通信路容量の近似値であると結論づける．

コスト 0 付近の値の近似が上手くいかなかった要因として，本来，ポアソン通信路の通信路容量を求めるためのものでない Blahut アルゴリズムを使用したことで計算の精度が低くなったことが考えられる．3 章の手順 3 で説明した遷移確率は，Blahut アルゴリズムにおいては遷移行列であった．ポアソン通信路の遷移確率には，階乗や指数関数が含まれており，アルゴリズムが正常に作動しなくなった可能性がある．また，入力確率分布の更新の際，式 (5) の計算が正しく行われずに更新量が微小になり，指定した反復回数 10,000 回以内で収束条件を満たすことができなくなったとも考えられる．

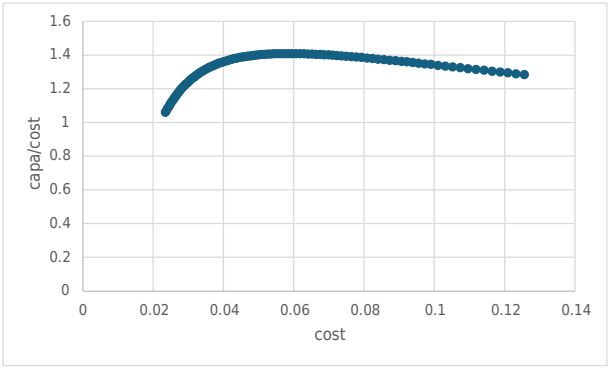

図 4　パラメータ $1.0 \leq s < 2.0$ のときのコストあたり通信路容量

5.2　先行研究との比較

先行研究では，通信路入力が指数分布に従うとした場合，ポアソン通信路のコストあたり通信路容量の下界として 0.423 ビット/エネルギーという値が数学的に明らかになっている [1]．本研究でより大きな 1.410 ビット/エネルギーが算出されたことから，指数分布よりも適切な入力分布があることが分かる．

5.3　最適な分布

本研究における通信路容量の最大値をとるときの入力分布を図 5 に，出力分布を図 6 に示す．入力分布も出力分布も，特定の x と y に偏っていることが分かる．入力分布は，グラフに表示されている $x = 1.337504$ の前後に，グラフには表示されていない非常に微小な確率を取っている．本研究においては，この入力分布の際にコストあたり通信路容量は最大値をとったため最適な入力分布であるとしたが，先述の通り，計算に不具合が生じた可能性があるため，より適切な入力分布があると考えられる．

6　まとめ

本研究では，ポアソン通信路のコストあたり通信路容量を Blahut アルゴリズムにて近似的な計算を行った．ポアソン通信路の入力信号を離散化することで，離散無記憶通信路の通信路容量を求めるためのものである Blahut アルゴリズムを使用することができた．通信路容量が最大値を取るときの入力分布と出力分布も同時に調べることで，最適な入力分布も明らかにすることができた．しかし，ポアソン通信路の特徴から，Blahut アルゴリズムに適応できてい

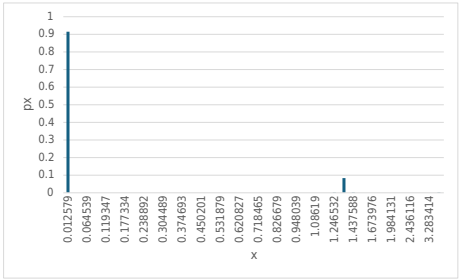

図 5　入力分布

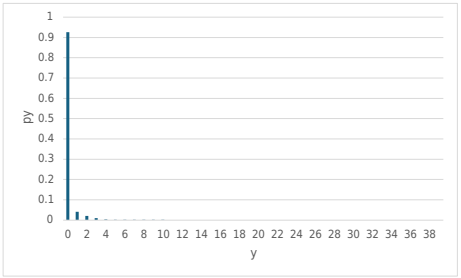

図 6　出力分布

ない部分もあることから，正確な近似ができたとは言えない．精度の向上のためには，不具合の原因を改善すること，よりポアソン通信路に適したアルゴリズムの検討が必要である．

謝辞

本研究を行うにあたり，丁寧なご指導を賜りました指導教員の西新幹彦准教授に深く感謝申し上げます．

参考文献

[1] 佐伯翼，「光子計数通信システムの通信路容量の下界について」，信州大学工学部卒業論文（指導教員：西新幹彦），2024 年 4 月．

[2] Richard E. Blahut, “Computation of Channel Capacity and Rate-Distortion Functions,” IEEE Transactions on Information Theory, vol. IT-18, no.4, pp.460-473, 1972.

付録 A　ポアソン通信路の通信路容量を求めるプログラム

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

#define INSIZE 40
#define OUTSIZE 40
#define LAMBDA 1.0
#define ERROR (1e-5)
#define MAX_ITER 10000 // 最大反復回数

// x の離散化
double realx(int i, int div) {
    if (i >= INSIZE) {
        fprintf(stderr, "[%d] Index out of bounds in realx: i = %d\n", __LINE__, i);
        exit(1);
    }
    return(-log(1.0 - (2.0 * i + 1.0) / (2.0 * div)) / LAMBDA);
}

// ポアソン通信路
double w(int y, double x) {
    double a = 1.0;
    for (int i = 0; i < y; i++) {
        a *= x / (i + 1);
    }
    return(a * exp(-x));
}

// gk(x)
double gk(double x) {
    double sum = 0.0;
    for (int y = 0; y < OUTSIZE; y++) {
        sum += w(y, x);
    }
    return(sum);
}

// コスト関数
double cost(double x) {
    return(x);
}

double arimoto(double s, double *capa, double *expc, double *output_px, double *output_py) {
    double px[INSIZE];
    double py[OUTSIZE];
    double IL = 0.0;
    double IU = ERROR * 2;

    // 初期化
    for (int i = 0; i < INSIZE; i++) {
        px[i] = 1.0 / INSIZE;
    }

    int iteration = 0;
    while (IU - IL > ERROR) {
        iteration++;
        if (iteration > MAX_ITER) {
            printf("計算を強制終了\n");
            break;
        }

        double cTmp[INSIZE];
        double sum = 0.0;
```

```
        // y の分布を更新
        for (int y = 0; y < OUTSIZE; y++) {
            py[y] = 0.0;
            for (int xi = 0; xi < INSIZE; xi++) {
                double x = realx(xi, INSIZE);
                py[y] += px[xi] * w(y, x) / gk(x);
            }
        }

        // 新しい px を計算
        for (int xi = 0; xi < INSIZE; xi++) {
            double x = realx(xi, INSIZE);
            double gkx = gk(x);
            if (gkx == 0.0) {
                fprintf(stderr, "gk(x) is zero for x = %lf\n", x);
                exit(1);
            }
            double cx = 0.0;
            for (int y = 0; y < OUTSIZE; y++) {
                double wyx = w(y, x);
                cx += wyx / gkx * log(wyx / gkx / py[y]);
            }
            cTmp[xi] = exp(cx - s * cost(x));
        }

        // IL, IU の更新
        double eIL = 0.0;
        double eIU = 0.0;
        for (int xi = 0; xi < INSIZE; xi++) {
            eIL += px[xi] * cTmp[xi];
            if (eIU < cTmp[xi]) {
                eIU = cTmp[xi];
            }
        }
        IL = log(eIL);
        IU = log(eIU);

        // 新しい px を正規化
        sum = 0.0;
        for (int xi = 0; xi < INSIZE; xi++) {
            px[xi] *= cTmp[xi];
            sum += px[xi];
        }
        for (int xi = 0; xi < INSIZE; xi++) {
            px[xi] /= sum;
        }
    }

    // px と py を呼び出し元に返す
    for (int xi = 0; xi < INSIZE; xi++) {
        output_px[xi] = px[xi];
    }
    for (int y = 0; y < OUTSIZE; y++) {
        output_py[y] = py[y];
    }

    // コストと相互情報量を計算
    *expc = 0.0;
    for (int xi = 0; xi < INSIZE; xi++) {
        *expc += px[xi] * cost(realx(xi, INSIZE));
    }
    *capa = s * *expc + IL;

    return *capa;
}

#define SADD 0.01
#define SMIN 0.0
```

```
#define SMAX 1.0

int main() {
    char *fname = "capa_cost.csv";
    char *px_fname = "max_px.csv";  // 最大容量時の入力分布を出力するファイル
    char *py_fname = "max_py.csv";  // 最大容量時の出力分布を出力するファイル

    double maxMutal = 0.0;
    double s = 0.0;
    double maxS = 0.0;

    double maxPx[INSIZE];  // 最大相互情報量時の入力分布を保存する配列
    double maxPy[OUTSIZE]; // 最大相互情報量時の出力分布を保存する配列
    double currentPx[INSIZE];
    double currentPy[OUTSIZE];

    FILE *fp = fopen(fname, "w");
    if (fp == NULL) {
        perror("ファイルが開けません");
        return -1;
    }

    fprintf(fp, "s,rcapa,rcost\n");

    for (double s = SMIN; s < SMAX; s += SADD) { //s の上限と下限を設定
        double capa, cost;
        printf("Running Arimoto with s = %lf\n", s);
        double tmpMutal = arimoto(s, &capa, &cost, currentPx, currentPy);

        fprintf(fp, "%lf,%lf,%lf\n", s, capa, cost);
        printf("%lf,%lf,%lf\n", s, capa, cost);

        if (tmpMutal > maxMutal) {
            maxMutal = tmpMutal;
            maxS = s;
            // 最大相互情報量時の px と py を保存
            for (int xi = 0; xi < INSIZE; xi++) {
                maxPx[xi] = currentPx[xi];
            }
            for (int y = 0; y < OUTSIZE; y++) {
                maxPy[y] = currentPy[y];
            }
        }
    }

    fclose(fp);

    // 最大相互情報量時の入力分布をファイルに出力
    FILE *px_fp = fopen(px_fname, "w");
    if (px_fp == NULL) {
        perror("px ファイルが開けません");
        return -1;
    }

    fprintf(px_fp, "x,px\n");
    for (int xi = 0; xi < INSIZE; xi++) {
        double x_value = realx(xi, INSIZE);  // xi を実際の x 値に変換
        fprintf(px_fp, "%lf,%lf\n", x_value, maxPx[xi]);
    }
    fclose(px_fp);

    // 最大相互情報量時の出力分布をファイルに出力
    FILE *py_fp = fopen(py_fname, "w");
    if (py_fp == NULL) {
        perror("py ファイルが開けません");
        return -1;
    }

    fprintf(py_fp, "y,py\n");
```

```
    for (int y = 0; y < OUTSIZE; y++) {
        fprintf(py_fp, "%d,%lf\n", y, maxPy[y]);
    }
    fclose(py_fp);

    return 0;
}
```