信州大学工学部

学士論文

# 整数の符号化を用いた<br>連続一様乱数の有歪み符号化について

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　21T2157E
氏名　山口 真凜

2025 年 3 月 11 日

# 目次

# 1　はじめに

情報化社会に拍車がかかる昨今，スマートフォンを始めとした通信機器を触らない日はないと言っても過言ではない．通信は情報源からデータを送る送信者とそのデータを受け取る受信者の間で行われる．情報源から送られるデータは符号器で符号化され符号語となり，受信者は受け取った符号語を復号器で復号することで元のデータに戻す．この時送信者と受信者のデータが完全に同一でなくても良い場合がある．例えば画像を送信する際，元画像だと認識できれば多少解像度が落ちた画像でも良い．このことは，情報源から得たデータと復号したデータがある程度の誤差を許容して通信を行ったと言い換えることができる．この許容した誤差を歪みと呼ぶ．一方，データ 1 シンボル当たりの符号語の長さを符号化レートと呼ぶ．符号化レートが小さいと効率の良い通信が可能だが，符号化レートを小さくすると歪みが大きくなるという関係がある．この関係を明らかにする問題をレート歪み問題という．

レート歪み問題における基本的な問題設定では，符号は歪みや符号化レートの値を固定した上で設計される．しかし画像を送受信する場合などでは，目的によって画像に求める解像度が違う場合がある．すなわち，送受信で求められる歪みの大きさが違う場合がある．しかし送受信で異なる歪みごとに異なる符号器を使うといったことはなく，我々は同一の符号器で色々な歪みの通信を行える．つまり，実際の通信で使用されている符号器は，歪みや符号化レートを柔軟に変えられる．特に復号器は復元したデータの歪みを知らずに復号する．本論文では実際の通信で用いられているような歪みを変化できる符号器を作成し，シミュレーションによってレートと歪みの関係を評価する．

# 2　定常無記憶情報源のレート・歪み理論

歪みを柔軟に変化させられる符号を考える前に，許容歪みを固定した従来の理論的成果を復習する．定常無記憶情報源を $\boldsymbol{X} = X^n = (X_1, X_2, \cdots, X_n)$ とし，情報源アルファベットを $\mathcal{X}$，復号アルファベットを $\mathcal{Y}$ とする．歪み測度 $d$ を関数 $d : \mathcal{X} \times \mathcal{Y} \to [0, +\infty)$ と定め，$d(x, y)$ を $x \in \mathcal{X}$ と $y \in \mathcal{Y}$ の間の歪みと呼ぶ．長さ $n$ のデータ $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ と $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ の歪み $d_n(\boldsymbol{x}, \boldsymbol{y})$ を

$$d_n(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} d(x_i, y_i) \tag{1}$$

と定義する．すると，情報源シンボル 1 個あたりの歪みは

$$\frac{1}{n} d_n(\boldsymbol{x}, \boldsymbol{y}) \geq 0 \tag{2}$$

で与えられる．

歪み測度 $d_n$ が与えられた際，情報源 $\boldsymbol{X}$ の符号化は次のように行う．$M_n$ 個の符号語の集合を $\mathcal{M}_n = \{1, 2, \cdots, M_n\}$ とする．そして，符号化を符号器 $\varphi_n : \mathcal{X}^n \to \mathcal{M}_n$ によって定め，復号化を復号器 $\psi_n : \mathcal{M}_n \to \mathcal{Y}^n$ によって定める．情報源の出力 $\boldsymbol{x} \in \mathcal{X}^n$ は符号器 $\varphi_n$ によって符号化された後，復号器 $\psi_n$ によって復元される．

次に定常無記憶情報源における達成可能性を定義する．

**定義 1**　レート $R$ が歪み測度 $d$ で $D$-達成可能とは

$$\limsup_{n\to\infty} \mathbb{E}\Big[\frac{1}{n} d_n(X^n, \psi_n(\varphi_n(X^n)))\Big] \leq D \tag{3}$$

$$\limsup_{n\to\infty} \frac{1}{n} \log M_n \leq R \tag{4}$$

を満たす符号の列 $\{(\varphi_n, \psi_n)\}_{\infty}^{n=1}$ が存在することである．

この定義は固定した $R$,$D$ に対して符号の存在を要請していることに注意されたい．固定した許容歪み $D$ に対して符号化レートをどこまで小さくできるのかという問題は，次の関数を明らかにする問題として定式化される．

**定義 2**　レート・歪み関数は以下のように定義される．

$$R_0(D) \triangleq \inf\{R | R \text{ が } D\text{-達成可能 }\} \tag{5}$$

$X$ と $Y$ の間の相互情報量 $I(X;Y)$ を

$$I(X;Y) \triangleq \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} P_{XY}(x,y) \log \frac{P_{Y|X}(y|x)}{P_Y(y)} \tag{6}$$

と定義すると，次の定理が成立することが知られている．

**定理 1** ([1])　分布 $P_X$ に従う定常無記憶情報源 $\boldsymbol{X}$ のレート・歪み関数は

$$R_0(D) = \min_{Y:\mathbb{E}[d(X,Y)]\leq D} I(X;Y) \tag{7}$$

で与えられる．

# 3　整数の符号化

本研究では情報源シンボルを一度整数に変換した後符号化を行う．整数の符号化としての複数の方法を検討する．本研究で採用した整数の符号化は，文献 [2] に基づき全て一意復号可能かつ可変長な符号となっている．

まずは単進符号である．アルファ符号とも呼ばれ，符号化したい整数値と同じ長さの符号語を割り当てる．1 以上の整数値 $m$ に対して単進符号の符号語長 $l_\alpha(m)$ は

$$l_\alpha(m) = m \tag{8}$$

と表される．

次にゴロム・ライス符号である．ゴロム・ライス符号は符号語長に影響する任意のパラメータが存在する．1 以上の整数値 $m$ に対してゴロム・ライス符号の符号語長 $l_{GR}(m)$ は整数値のパラメータ $d$ を用いて

$$l_{GR}(m) = \left\lfloor \frac{m}{2^d} \right\rfloor + d \tag{9}$$

と表される．

最後にガンマ符号，デルタ符号，オメガ符号である．この三つの符号はどれもピーター・イライアスによって開発された符号である．1 以上の整数値 $m$ に対して $\lambda(m) = \lfloor \log_2 m \rfloor$，$\lambda^k(m) = \lambda(\lambda^{k-1}(m))$ とする．ガンマ符号，デルタ符号，オメガ符号の符号語長をそれぞれ $l_\gamma(m)$，$l_\delta(m)$，$l_\omega(m)$ とすると，順に

$$l_\gamma(m) = 2\lambda(m) + 1 \tag{10}$$

$$l_\delta(m) = 2\lambda(\lambda(m) + 1) + \lambda(m) + 1 \tag{11}$$

$$l_\omega(m) = \sum_{i=1}^{k} \lambda^i(m) + k + 1 \quad (\lambda^k(m) = 1) \tag{12}$$

と表される．

## 4　許容最大歪みの変更に対応した符号化

本研究では情報源アルファベット $\mathcal{X}$ と復号アルファベット $\mathcal{Y}$ は共に [0,1] の単位区間とし，情報源の分布は [0,1] 上の一様分布とする．歪み関数 $d : \mathcal{X} \times \mathcal{Y} \to [0, +\infty)$ は具体的に $d(x, y) \triangleq |x - y|$ とする．復号器は許容最大歪み $D$ と情報源系列 $\mathcal{X}$ に対してその符号語を出力する．復号器は受け取った符号語のみから $\frac{1}{n} d_n(\boldsymbol{x}, \boldsymbol{y}) \leq D$ となる $\boldsymbol{y}$ を出力する．このような符号器と復号器を具体的に以下のように実現する．

本研究におけるエンコーダのアルゴリズムについて述べる．まず情報源のシンボルとして乱数値を取得する．この乱数値を基に，近似値の候補の中から許容最大歪み以下である値を近似値として選択する．次に近似値を対応する整数値へ変換する．最後にその整数値を 3 章で述べた整数の符号化で符号化する．以下では各フェーズについて詳しく説明する．

近似値の候補について説明する．先述したように，本研究では 0.0 以上 1.0 未満の乱数値をデータとして得るため，図 1 のような 0.0 から 1.0 の範囲の数直線を考えたい．許容最大歪みが 0.5 以上の場合，乱数値がどの値を取っても 0.5 に近似できるとわかる．このため，近似値の候補として 0.5 が存在する．また，許容最大歪みが 0.25 以上 0.5 未満の場合，乱数値がどのような値を得ても 0.25，0.5，0.75 のいずれかに近似できる．したがって先ほどの 0.5 に加えて，0.25，0.75 も近似値の候補とする．このように歪みが小さい場合を考えることで近似値の

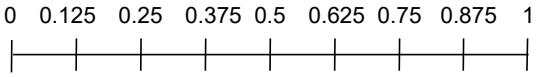


図 1　[0.0，1.0) の範囲の数直線

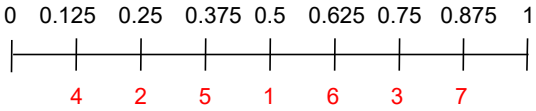


図 2　近似値と整数値の対応

候補を無限に増やした．例えば許容最大歪みが 0.125 以上 0.25 未満の場合，近似値の候補は図 1 のようになる．

近似値を整数値に変換する操作ついて説明する．許容最大歪みが 0.5 以上の場合はどれだけ歪みがあっても 0.5 に近似される．さらに歪みが小さくなった場合にも近似値の候補に 0.5 が存在し続ける．そのため 0.5 に近似される確率が高いと考えた．整数の符号化では整数値が小さいほど符号語長が短いため，近似される確率が高い近似値により小さな整数値を対応させる方が適切だと考えた．したがって近似値 0.5 に整数値 1 を与えた．同様に，歪みが大きい場合に近似できる候補値ほど小さい整数値を与えるとした．図 1 の近似値の候補に，歪みが大きい順に小さな整数値を振ると図 2 のようになる．0.25 と 0.75 のように，同じ大きさの歪みで近似可能な候補値について，本研究では 0.0 に近い値から順に小さい整数値を当てる．

このようなエンコーダのレート・歪み関数を求める．アルゴリズムを用いてデータを整数値に変換した後，整数値を符号化することでレートを，復号したデータと元データを比較することで歪みを求めた．許容最大歪みは 0.1 から 0.49 までを 0.1 ずつ動かし，回数は各歪みごとに十万回とした．またゴロム・ライス符号のパラメータは 2 とした．

## 4.1　1 シンボルごとの符号化

上記のアルゴリズムを用いて 1 シンボルごとに符号化するエンコーダのレート・歪み関数を求めた．図 3 はシミュレーションの結果と理想的なレート・歪み関数を表したグラフである．横軸が歪み，縦軸が符号化レートを指す．シミュレーション結果が理想的なレート・歪み関数のグラフよりも大きく上に位置している．単進符号，ガンマ符号，デルタ符号，オメガ符号の 4 つは符号化レートが 1 に収束しており，ゴロム・ライス符号は 2 に収束している．歪みをさらに 0 へ近づけていくと，オメガ，デルタ，ガンマ，ゴロム・ライス，単進符号の順に符号化レートが小さくなっている．

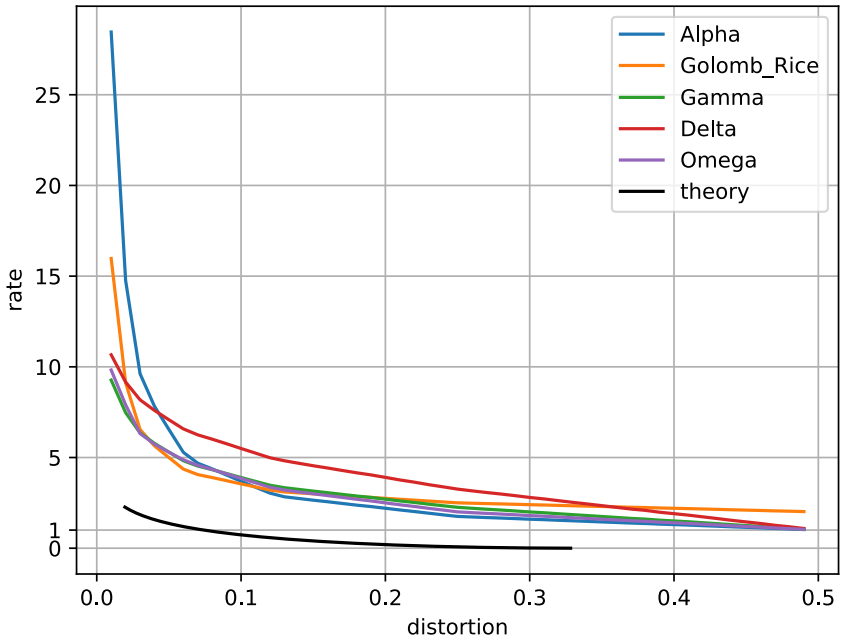


図 3　1 シンボルごとに符号化したレート・歪み関数

3 章で述べた各符号の符号語長から，単進符号の符号化語長が最も大きくなる．しかし図 3 では，歪みが小さい範囲を除くと，本研究で比較した符号の中でデルタ符号の符号化レートが大きくなっている．歪みが大きな範囲では小さい整数値を符号化することになるが，デルタ符号は大きな整数値を符号化するのに適した符号化方法である [2]．そのため，デルタ符号の長所が生かされず符号化レートが大きくなったと考えた．

## 4.2　2 シンボルごとの符号化

シミュレーションの結果をより理想的なレート歪み関数に近づけるためには，符号化レートをさらに小さくする必要がある．そこで，複数シンボルまとめて符号化をすることで符号化レートを抑えられると予想した．例えば 2 シンボルごとに符号化すると 1 シンボル当たりの符号語長が送信符号の $\frac{1}{2}$ に抑えられると考えた．

2 シンボルごとに符号化するエンコーダのアルゴリズムを述べる．まず 1 シンボルごとに符号化するアルゴリズムを利用して，乱数値 2 つを近似し整数値にそれぞれ変換する．得られた 2 つの整数値を $m_1$,$m_2$ とした．そして表 1 のように組 $(m_1,m_2)$ から整数値 1 つを決定した．組 $(m_1,m_2)$ で決まる値をグループに分け，各グループごとに整数値を与えた．この整数値に対して符号化を行う．

表 1　整数値 2 つから整数値 1 つを決める表

| | | $m_1$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... |
| | 1 | 1 | 4 | 7 | 22 | 29 | 36 | 43 | 116 | |
| | 2 | 2 | 5 | 8 | 23 | 30 | 37 | 44 | 117 | |
| | 3 | 3 | 6 | 9 | 24 | 31 | 38 | 45 | 118 | |
| $m_2$ | 4 | 10 | 14 | 18 | 25 | 32 | 39 | 46 | 119 | ... |
| | 5 | 11 | 15 | 19 | 26 | 33 | 40 | 47 | 120 | |
| | 6 | 12 | 16 | 20 | 27 | 34 | 41 | 48 | 121 | |
| | 7 | 13 | 17 | 21 | 28 | 35 | 42 | 49 | 122 | |
| | 8 | 50 | 58 | 66 | 74 | 82 | 90 | 98 | 123 | |
| | ⋮ | | | | ⋮ | | | | | |

上記の方法でエンコーダのレート・歪み関数をそれぞれ求めた．図 4，5 がシミュレーションの結果と 1 シンボルごとに符号化したレート・歪み関数のグラフである．グラフ (a) がアルファ，ゴロムライス符号，グラフ (b) がデルタ，ガンマ，オメガ符号のグラフであり，横軸が歪み，縦軸が符号化レートを指す．凡例に *two_* と付いているのが 2 シンボルごとに符号化したレート歪み関数のグラフである．2 シンボルずつ符号化したグラフは 1 シンボルずつ符号化したグラフに比べて符号化レートがあまり小さくなっておらず，改善しているとはいえない．

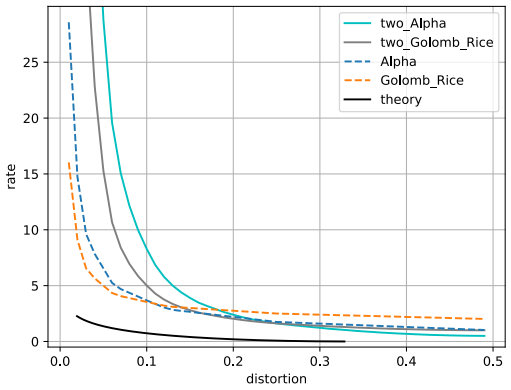


図 4　2 シンボルごとに符号化したレート・歪み関数 (a)

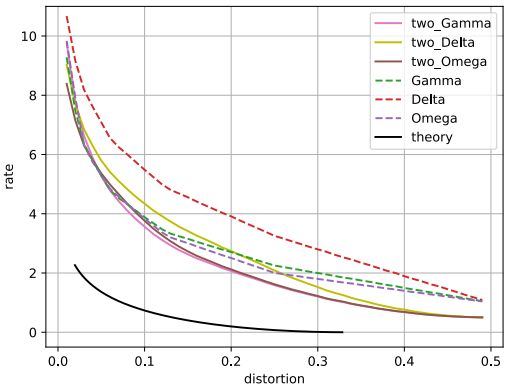


図 5　2 シンボルごとに符号化したレート・歪み関数 (b)

## 4.3 改善後の２シンボルごとの符号化

2 シンボルごとに符号化するアルゴリズムをさらに改善する方法を考える．4.2 節で述べたアルゴリズムでは，2 つの乱数値を 2 つの整数値に変換し，そこから 1 つの整数値を決定し符号化している．2 つの乱数値を 2 つの整数値に変換するまでの操作は 1 シンボルごとに符号化する場合と同じ操作を 2 回行っているだけであるため，2 つの整数値から 1 つの整数値を決定する操作を見直すことにした．

ここで，表 2 のような整数値の与え方を考え，表 1 と同じグループに分けた．しかし，表 1 で分けられていたグループをさらに 2 つに分け，各グループにおいて整数値の与え方を変化させた．整数値は，グループ内で網掛けの部分から先に小さい整数値を与えた．歪みを抑えるにつれ，$m_1$，$m_2$ の取りえる候補数が指数的に増えている．そのため各グループ内で許容最大歪みの小さい $m_1$，$m_2$ から決まる整数値が多い．したがって近似値の候補を採用する際に，許容最大歪みを抑えた候補同士から決まる値，つまり網掛けの値を選ぶ確率がグループ内で高くなるのではと考えた．

上記のアルゴリズムを用いて 2 シンボルごとに符号化したエンコーダのレート・歪み関数を再度計測した．図 6，7 がシミュレーションの結果と 1 シンボルごとに符号化したエンコーダのレート・歪み関数のグラフである．グラフ (a) がアルファ，ゴロムライス符号，グラフ (b) がデルタ，ガンマ，オメガ符号のグラフだ．凡例に $two_$と付いている 2 シンボルごとに符号化したグラフを図 4，5 と図 6，7 で比較すると，図 6，7 のグラフの方が同じ歪みの場合に符号化レートが小さくなっているとわかる．また図 6，7 においてガンマ符号，デルタ符号，オ

表 2　改善後の整数値 2 つから整数値 1 つを決める表

| | | $m_1$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... |
| $m_2$ | 1 | 1 | 8 | 9 | 38 | 41 | 44 | 47 | 170 | |
| | 2 | 6 | 2 | 4 | 39 | 42 | 45 | 48 | 171 | |
| | 3 | 7 | 3 | 5 | 40 | 43 | 46 | 49 | 172 | |
| | 4 | 26 | 30 | 34 | 10 | 14 | 18 | 22 | 173 | |
| | 5 | 27 | 31 | 35 | 11 | 15 | 19 | 23 | 174 | ... |
| | 6 | 28 | 32 | 36 | 12 | 16 | 20 | 24 | 175 | |
| | 7 | 29 | 33 | 37 | 13 | 17 | 21 | 25 | 176 | |
| | 8 | 114 | 122 | 130 | 138 | 146 | 154 | 162 | 50 | |
| | ⋮ | | | | ⋮ | | | | | |

メガ符号においては 2 シンボルごとに符号化を行ったレート・歪み関数の方が，1 シンボルごとに符号化を行うレート・歪み関数よりも特に優れている．しかし，単進符号やゴロム・ライス符号では歪みが小さい場合において 2 シンボルごとに符号化を行うよりも 1 シンボルごとに符号化を行ったレート・歪みグラフの方が優れている．

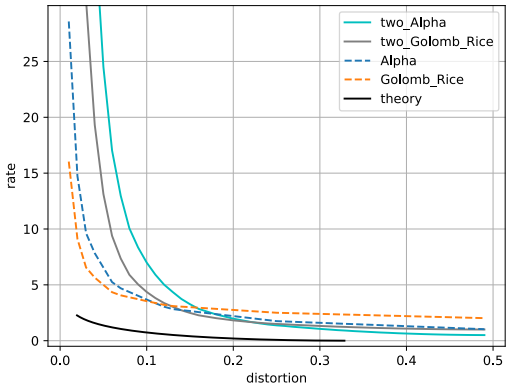


図 6　改善後の 2 シンボルごとに符号化したレート・歪み関数 (a)

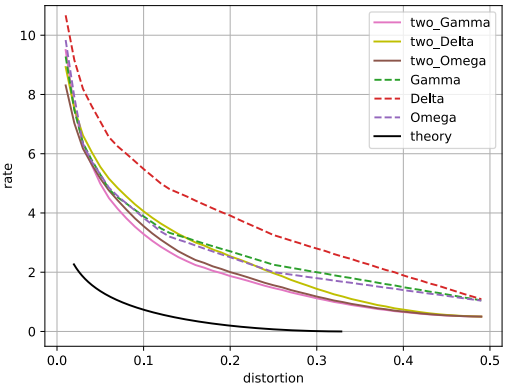


図 7　改善後の 2 シンボルごとに符号化したレート・歪み関数 (b)

これらの結果から近似値と整数値の対応が適切であれば，複数シンボル同時に符号化を行う方がそれぞれの符号化方法でより良いレート・歪み関数になることが確認できた．また，2 シンボルごとに符号化する場合，符号語長を整数値 $\div 2$ で計算することで 1 シンボルごとに符号化するよりも符号化レートを抑えていたが，歪みが小さい場合においては整数値が膨大な値になる．したがって膨大な整数値を 2 で割る程度では大きさを抑えられなくなり，単進符号，ゴロム・ライス符号においては特に，歪みが小さい範囲において 1 シンボルごとに符号化をするレート・歪み関数の方が優れたグラフになったと考えられる．

近似値と整数値の対応を見直すにあたって，本研究では近似値の候補の増え方に着目した．しかし，レート歪み関数を改善するための別の方法として二値の範囲を軸とした二次元平面の距離を利用する方法も考えた．例えばボロノイ図を用いる方法がある．ボロノイ図とは同一空間で点同士の距離で領域を分けた図で，特に二次元平面では領域の境界線が各点の二等分線になる．このように点同士の距離から近似値に整数値を振る方法でもレート・歪み関数が改善できると考えている．

## 5　まとめ

本研究では，符号化の際に異なる歪みに対応できる符号器を作成した．その符号器を用いて定常無記憶通信路でレート・歪み関数を算出し，理想的なレート歪み関数と比較した．1 シン

ボルごとに符号化を行う場合よりも 2 シンボルごとに符号化を行う場合の方が理想的なレート・歪み関数により近いグラフになっていた．今後の課題として，本研究で扱った符号器を数理モデルを用いて再評価することが挙げられる．数学的に証明を行うことで理論的な観点から理想的なレート・歪み関数により近づけていきたい．また，本研究で用いた符号化方法よりもさらに理想的な符号化方法を研究，評価していくことも検討している．

## 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感謝の意を表する．

## 参考文献

[1] 韓太舜，情報理論における情報スペクトル的方法，培風館，1998.

[2] 韓太舜，小林欣吾，情報と符号化の数理，岩波書店，1994.

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

[4] 澤田真子,「観測値に雑音のあるシステムのレート・歪み関数の数値計算と基礎的考察」, 信州大学工学部卒業論文 (指導教員：西新幹彦)，2020 年 2 月.

[5] 渡辺一帆,「レート歪み関数と最適再構成分布」, https://www.ieice.org/ess/sita/forum/article/2019/201909050913.pdf, 2024 年 12 月閲覧.

[6] Peter Elias, “Universal Codeword Sets and Representations of the Integers,” IEEE Transactions on Information Theory, Vol. IT-21, no.2, pp.194–203, Mar. 1975.

# 付録 A　ソースコード

## A.1　1 シンボルごとに符号化するアルゴリズムのプログラム

```
from re import I
import numpy as np
import math
import matplotlib.pyplot as plt
import random
%matplotlib inline
kaisu: int =100000
gosa = np.arange(1,50)/100
avg_tansin_hugogo=[]
avg_gr_hugogo=[]
avg_gamma_hugogo=[]
avg_delta_hugogo=[]
avg_omega_hugogo=[]

for g in range (len(gosa)):
    tansin_hugogo=[]
    gr_hugogo=[]
    gamma_hugogo=[]
    delta_hugogo=[]
    omega_hugogo=[]
    for i in range (kaisu):
        x: float =np.random.random()
        y: float =0.5
        tansin_hugo: int =0
        gr_hugo: int =0
        gamma_hugo: int =0
        delta_hugo: int =0
        omega_hugo: int =0
        seisu: int
        d: int =1
        h: int =0
        d_gr: int =2
        while True:
            #乱数値に対する各近似値の探索
            if abs(x-y)>gosa[g]:
                d+=1
                if x<y:
                    y=y-(1/(2**d))
                elif x>y:
                    y=y+(1/(2**d))
            elif abs(x-y)<=gosa[g]:
                #近似値を整数値に変換
                for k in range(2**(d-1)):
                    if ((2*(k+1))-1)/(2**d)==y:
                        seisu=(k+1)+(2**(d-1))+(-1)
                        tansin_hugo=seisu
                        tansin_hugogo.append(tansin_hugo)
                        gr_hugo=math.floor(seisu/d_gr)+d_gr
                        gr_hugogo.append(gr_hugo)
                        gamma_hugo=math.floor(2*math.log2(seisu))+1
                        gamma_hugogo.append(gamma_hugo)
                        delta_hugo=math.floor(2*math.log2(math.floor(2*math.log2(seisu))+1))\
                        +math.floor(math.log2(seisu))+1
                        delta_hugogo.append(delta_hugo)
                        while True:
                            if seisu==1:
                                omega_hugo+=h+1
                                break
                            omega_hugo+=math.floor(math.log2(seisu))
                            seisu=math.floor(math.log2(seisu))
                            h+=1
```

```
                        omega_hugogo.append(omega_hugo)
                        break
                break
    avg_tansin_hugogo.append(sum(tansin_hugogo)/kaisu)
    avg_gr_hugogo.append(sum(gr_hugogo)/kaisu)
    avg_gamma_hugogo.append(sum(gamma_hugogo)/kaisu)
    avg_delta_hugogo.append(sum(delta_hugogo)/kaisu)
    avg_omega_hugogo.append(sum(omega_hugogo)/kaisu)

plt.figure()
plt.plot(gosa, avg_tansin_hugogo,label='Alpha')
plt.plot(gosa, avg_gr_hugogo,label='Golomb_Rice')
plt.plot(gosa, avg_gamma_hugogo,label='Gamma')
plt.plot(gosa, avg_delta_hugogo,label='Delta')
plt.plot(gosa, avg_omega_hugogo,label='Omega')
plt.grid()
plt.legend()
plt.xlabel("distortion")
plt.ylabel("rate")
plt.yticks([0,1,5,10,15,20,25,30])
```

## A.2　2 シンボルごとに符号化するアルゴリズムのプログラム

```
from re import I
import numpy as np
import math
import matplotlib.pyplot as plt
import random
%matplotlib inline
kaisu: int =100000
gosa = np.arange(1,50)/100
avg_tansin_hugogo=[]
avg_gr_hugogo=[]
avg_gamma_hugogo=[]
avg_delta_hugogo=[]
avg_omega_hugogo=[]

def cut(a,b,c,d):
    if abs(a-b)>c:
        if a<b:
            b=b-(1/(2**d))
        elif a>b:
            b=b+(1/(2**d))
    return b

for g in range (len(gosa)):
    tansin_hugogo=[]
    gr_hugogo=[]
    gamma_hugogo=[]
    delta_hugogo=[]
    omega_hugogo=[]
    for i in range (kaisu):
        x: float =np.random.random()
        y: float =np.random.random()
        cx: float =0.5
        cy: float =0.5
        kx: int =0
        ky: int =0
        tansin_hugo: int =0
        gr_hugo: int =0
        gamma_hugo: int =0
        delta_hugo: int =0
        omega_hugo: int =0
        seisu: int
        d: int =1
```

```
        h: int =0
        d_gr: int =2
        while True:
            #乱数値に対する各近似値の探索
            if (abs(x-cx)+abs(y-cy))/2>gosa[g]:
                d+=1
                if abs(x-cx)>=abs(y-cy):
                    if x<cx:
                        cx=cx-(1/(2**d))
                        cy=cut(y,cy,gosa[g],d)
                    elif x>cx:
                        cx=cx+(1/(2**d))
                        cy=cut(y,cy,gosa[g],d)
                elif abs(x-cx)<abs(y-cy):
                    if y<cy:
                        cy=cy-(1/(2**d))
                        cx=cut(x,cx,gosa[g],d)
                    elif y>cy:
                        cy=cy+(1/(2**d))
                        cx=cut(x,cx,gosa[g],d)
            elif (abs(x-cx)+abs(y-cy))/2<=gosa[g]:
                #各近似値を整数値に変換
                for dk in range(d):
                    for k in range(2**dk):
                        if ((2*k)+1)/(2**(dk+1))==cx:
                            kx=k+(2**dk)
                        if ((2*k)+1)/(2**(dk+1))==cy:
                            ky=k+(2**dk)
                        if (kx!=0)and(ky!=0):
                            break
                    else:
                        continue
                    break
                #各整数値から送信に用いる値の決定
                if (kx>=2**(d-1))and(ky>=2**(d-1)):
                    seisu=-3*(2**(d-1))+((2**(d-1))*kx)+ky+2
                elif (kx<2**(d-1))or(ky<2**(d-1)):
                    if (kx<ky):
                        seisu=(2**(d-1))*(kx-4+(2**d))+ky+2
                    elif (ky<kx):
                        seisu=(2**(d-1))*(ky-5+(2**d))+kx+2
                tansin_hugo=seisu
                tansin_hugogo.append(tansin_hugo)
                gr_hugo=math.floor(seisu/d_gr)+d_gr
                gr_hugogo.append(gr_hugo)
                gamma_hugo=math.floor(2*math.log2(seisu))+1
                gamma_hugogo.append(gamma_hugo)
                delta_hugo=math.floor(2*math.log2(math.floor(2*math.log2(seisu))+1))+math.floor(math.log2(seisu))+1
                delta_hugogo.append(delta_hugo)
                while True:
                    if seisu==1:
                        omega_hugo+=h+1
                        break
                    omega_hugo+=math.floor(math.log2(seisu))
                    seisu=math.floor(math.log2(seisu))
                    h+=1
                omega_hugogo.append(omega_hugo)
                break
    avg_tansin_hugogo.append(sum(tansin_hugogo)/(2*kaisu))
    avg_gr_hugogo.append(sum(gr_hugogo)/(2*kaisu))
    avg_gamma_hugogo.append(sum(gamma_hugogo)/(2*kaisu))
    avg_delta_hugogo.append(sum(delta_hugogo)/(2*kaisu))
    avg_omega_hugogo.append(sum(omega_hugogo)/(2*kaisu))

plt.figure()
#plt.plot(gosa, avg_tansin_hugogo,label='Alpha')
#plt.plot(gosa, avg_gr_hugogo,label='Golomb_Rice')
plt.plot(gosa, avg_gamma_hugogo,label='Gamma')
plt.plot(gosa, avg_delta_hugogo,label='Delta')
```

```
plt.plot(gosa, avg_omega_hugogo,label='Omega')
plt.grid()
plt.legend()
plt.xlabel("error")
plt.ylabel("avg_length of code")
```

## A.3 理想的なレート・歪み関数を算出するプログラム

```
from re import I
import numpy as np
import math
import matplotlib.pyplot as plt
import random
RESO=1024
def ddd(xi,yi):
x:float =xi/RESO
y:float =yi/RESO
return(abs(x-y))
s: float
def kansu(s):
a: float = [[0]*RESO for i in range(RESO)]
for j in range(0, RESO, 1):
for k in range(0, RESO, 1):
a[j][k] = math.e**(s*ddd(j,k))
p: float =[1/RESO]*RESO
q: float = [1/RESO] * RESO
c: float = [0] * RESO
tu: float =0
tl: float =1
e: float
while (abs(tu-tl)<10**(-6)):
b:float = [0]*RESO
#c[y] の分母 b[x]
for j in range(0, RESO, 1):
for k in range(0, RESO, 1):
b[j] += q[k] * a[j][k]
#c[y] の計算
c=[0]*RESO
for k in range(0, RESO, 1):
for j in range(0, RESO, 1):
c[k] += p[j] * a[j][k]/b[j]
#q[y] と Tu,Tl の計算
tu=0
for k in range(0, RESO, 1):
q[k]= q[k]*c[k]
tu += q[k] * math.log(c[k])
tl=math.log2(max(c))
Q =[[0]*RESO for i in range(RESO)]
D = 0
R = 0
#Q[k|j] の分母 b2[j]
b2= [0]*RESO
for j in range(0, RESO, 1):
for k in range(0, RESO, 1):
b2[j] += a[j][k]*q[k]
#Q[k|j] の計算
for k in range(0,RESO, 1):
for j in range(0, RESO, 1):
Q[k][j] = a[j][k]*q[k]/b2[j]
#D の計算

for j in range(0, RESO, 1):
for k in range(0, RESO, 1):
D += p[j]*Q[k][j]*ddd(j,k)
#R(D) の計算
```

```
b3: float = 0
for j in range(0, RESO, 1):
b3 += p[j]*(math.log(b2[j]))
R = s*D - b3- tu
return D,R
#実装部
s=-500
n=0
D_list=[]
RD_list=[]
while n<500:
d,rd=kansu(s/10)
s+=1
n+=1
D_list.append(d)
RD_list.append(rd)
plt.figure()
plt.plot(D_list, RD_list)
plt.grid()
plt.xlabel("distortion")
plt.ylabel("rate")
```