信州大学工学部

学士論文

情報量を最大化する符号化方法の検討

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　16T2155J
氏名　三島 聖冬

2020 年 2 月 26 日

目次

1　はじめに

近年の自然災害の一つに活火山の噴火や河川の氾濫が挙げられる． 噴煙による灰の被害や家の浸水など私たちの生活に甚大な影響を与える災害は，今後も警戒しなければならない．そのため，そのような災害を私たちに伝えるシステムを考えることが必要になる．

例えば，活火山の噴火を知らせるシステムを考える．このシステムを構築するために重要となるものが「情報量」である．活火山の噴火といえども噴火という事象は頻繁に起こるわけではない．噴火という事象の起こる確率はとても小さいものであるが，私たちが知れば驚く事象であることは間違いない． 逆に噴火していない事象の起こる確率は大きいが，私たちが知っても全く驚かないであろう．情報量とは，情報が私たちに伝わった際にどれくらい驚くかという尺度に帰着できる．すなわち，活火山が噴火するという事象は情報量が大きく，噴火していないという事象は情報量が小さくなる．このことから, 情報量とその事象が起こる確率はトレードオフの関係になることが直感的にわかるであろう．

このシステムを構築する際に，火山が噴火していないという事象を私たちに伝えるということは無駄である．私たちが欲しい情報は，活火山が噴火したという情報である．つまり，情報源アルファベットから出力されるシンボルで情報量を多く持つ，言い換えると生起確率の小さいシンボルを優先して符号化していくことが必要になる．

実際に使われている符号化方法の一つに固定長符号化と呼ばれるものがある．固定長符号化は, 情報源アルファベットが出力されるシンボルに割り当てられる符号語をすべて同じ長さにする符号化方法である．この符号化方法では，情報源シンボルに割り当てる符号語数を少なくすることによって符号化の効率をよくできる．しかし，符号語数を少なくするということは，情報源から出力されるすべてのシンボルを正しく符号化できる訳ではない．通常の固定長符号化では，シンボルのうち生起確率の小さいものを正しく符号化しないことで情報が正確に伝わる確率を大きくしている．すなわち，情報量の大きいシンボルを正しく符号化していない符号化方法である．

本研究では雑音のない通信路モデルを考え，情報源から出力されるシンボルの生起確率の小さい，すなわち情報量の大きいシンボルを優先的に符号化し，受信側でエラー，ビットレート，情報量の観点から従来使われている固定長符号化と本研究の符号化を比較し符号化の特徴を検証していく．

2　情報量を最大化する符号化

2.1　固定長符号化

符号化の方法には，可変長符号化と固定長符号化が存在するが，本研究と関係が深い固定長符号化について説明する．

情報源アルファベットを $\mathcal{X}$ とする．その情報源アルファベットの中のシンボル x が生起する確率を，

$$P_X(x), x \in \mathcal{X} \tag{1}$$

と表す．このとき

$$I(x) = -\log_2 P_X(x) \tag{2}$$

と定義し，x の自己情報量と呼ぶ．また，

$$H(X) = -\sum_{i=1}^{n} P_X(x_i) \log_2 P_X(x_i) \tag{3}$$

と定義し，情報源 X のエントロピーと呼ぶ．

情報源から出力される情報源シンボルを n 文字ずつ区切る．新たに作られる拡大シンボル x^n は，$x^n = x_1 x_2 ... x_n \in \mathcal{X}^n$ と表される．

n 文字ずつ区切った情報源シンボルの生起確率は，それ以前に出力された情報源シンボルに依存せず，定常無記憶であるため，

$$P_{X^n}(x^n) = \prod_{i=1}^{n} P_X(x_i) \tag{4}$$

で表される．このように新たに生成された情報源シンボルを作り出す操作をアルファベット拡大という．

固定長符号化では，情報源シンボルを符号化する際に同じ長さの符号語で符号化を行う．符号語の個数を M_n とする．必要となる符号語の長さは

$$\lceil \log_2 M_n \rceil \ [bit] \tag{5}$$

となる．$\log_2 M_n$ が小数点をとる場合，繰り上げられた整数値となる．

情報源シンボルを n 文字ごとに符号化しているため，1 シンボルあたりのビット数であるビットレートは，

$$R = \frac{1}{n}\lceil \log_2 M_n \rceil \ [bit/symbol] \tag{6}$$

となる．

2.2 誤り確率を最小化する符号化

固定長符号化において, ビットレートを下げることを考える．ビットレートを下げるため，すべての情報源シンボルを正しく符号化することはあきらめ，符号語数はシンボルの数より少なくする．それでも尚，情報が正しく届く確率を大きくしたい．そこで情報源シンボルの誤り確率が小さくなるように，生起確率の大きい情報源シンボルを正しく復号していく．具体的には，情報源シンボルの生起確率の大きい方から符号語を割り当て，生起確率の小さい情報源シンボルは，他に割り当てられていた符号語に適当に割り当てる．そのとき，符号語を適当に割り当てられたシンボルは，受信側で復号化されるが，元のシンボルには復号されない．このことを復号誤りと呼び，復号誤りを起こす確率のことを誤り確率と呼ぶ．図 1 では情報源アルファベット D と E が復号誤りを起こすことになり，その誤り確率は情報源アルファベットから D と E が出力される生起確率になる．

誤り確率を最小化するこの符号化方法を本研究では，「誤り確率を最小化する符号化」と呼ぶことにする．

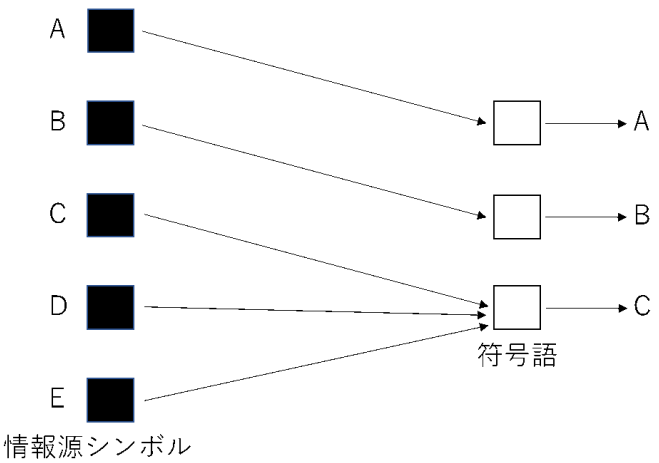

図 1　誤り確率を最大化する符号化のイメージ図

2.3　情報量を最大化する符号化

拡大シンボル x^n を考える．先ほどと同じようにビットレートを下げたいが受信側に届く情報量をできるだけ大きくするようにしたい．符号語を持つ拡大シンボルが受信側にもたらす情報量は，$-\log_2 P_{X^n}(x^n)$ である．符号語を持たず, 破棄された拡大シンボルの情報量は 0 となる．受信側に届く情報量は，符号語を持つ拡大シンボルの自己情報量を足した和であるため，符号語を持つ拡大シンボルの個数を k 個とすると，$-\sum_{i=1}^{k} \log_2 P_{X^n}(x^n)$ となる．この受信側に届く情報量を最大化するためには，できるだけ情報量の大きいほうからシンボルを符号化していく必要がある．

このシンボルの生起確率の小さいものから符号語を割り当て，余ったシンボルを破棄する符号化方法を「情報量を最大化する符号化」と呼ぶことにする．

2.2 節の「誤り確率を最小化する符号化」では，不要なシンボルを適当に他の符号語に割り当てていたが，「情報量を最大化する符号化」では，不要なシンボルを破棄している．情報量を最大化する符号化を行う際に，もし不要なシンボルを他の符号語に適当に割り当ててしまうと受信側で正確な情報量を測ることができないため，不要なシンボルを破棄している．

また，情報源シンボルが破棄されることを此処では，エラーと呼ぶことにする．

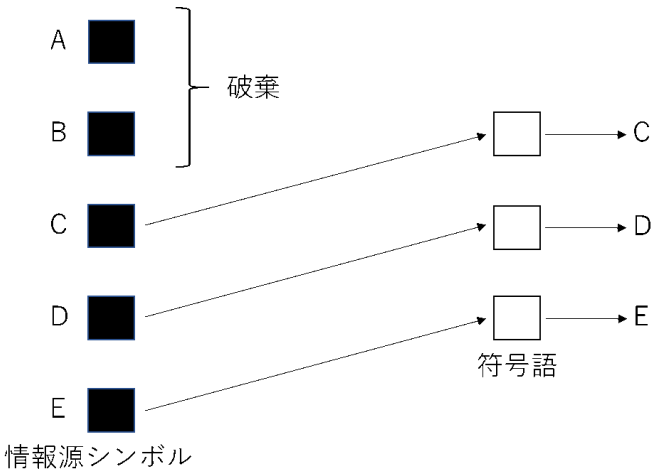

図 2　情報量を最大化する符号化のイメージ図

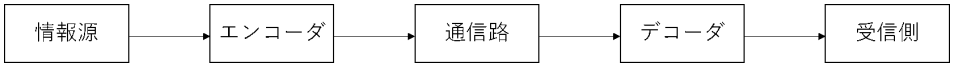

図 3　本研究における通信路モデル

2.4　研究目的

誤り確率を最小化する符号化では，ビットレートを大きくすると誤り確率は小さくなる．ビットレートが $H(X)$ より大きいと，情報源シンボルを区切るサイズであるブロック長 n が ∞ のとき，誤り確率が 0 になることが文献 [1] より知られている．

一方，情報量を最大化する符号化ではビットレートを大きくすると破棄される確率が小さくなり，受信側に届く情報量は大きくなることは自明だが，ビットレートを固定してブロック長を ∞ とするとき，エラーや届く情報量の振る舞いは，よく知られていない．

本研究では，情報量を最大化する符号化についてビットレート，エラー，届く情報量の振る舞いや相互の関係を数値計算によって明らかにする．

3　準備

3.1　雑音のない通信路

本研究の伝送モデルを説明する．一般的な通信路モデルと違い符号化された情報が通信路を通る際に雑音が加わらない，つまり符号語が通信路上で変わることがない通信路を考える．このような通信路モデルを図 3 に示す．送信側でビットレートを固定したもとで受信側でビットレート，エラー，情報量の尺度から符号化の特徴を調べる．誤り確率を最小化する符号化を行う際には，ビットレートを固定したときに符号化しない情報源シンボルを他のシンボルに割り当てる．情報量を最大化する符号化を行う際には，符号化しない情報源シンボルは送信側ですべて破棄される．

3.2　小さい情報源における符号化の性能

ブロック長を大きくしたときの振る舞いを調べる前に，以降の準備として，アルファベット拡大をしない情報源を例に挙げて符号化した時の振る舞いを調べる．表 1 のような情報源シンボルと出現確率が与えられたとする．

情報源シンボルが 5 つ存在し，それらにそれぞれ出現確率 $P_X(x)$ が与えられている．初めに, 誤り確率を最小化する符号化方法と情報量を最大化する符号化方法のビットレートとエ

ラーの関係を図 4，図 5 に示す．

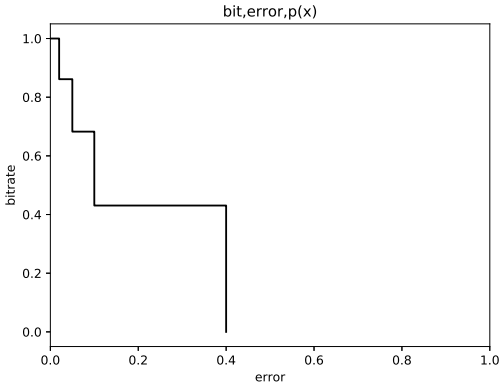

図 4　誤り確率を最小化する符号化を行った際のビットレートとエラーの関係

情報源シンボル	生起確率
A	0.6
B	0.3
C	0.05
D	0.03
E	0.02

表 1　情報源アルファベットから出力されるシンボルと出てくる確率 $P_X(x)$

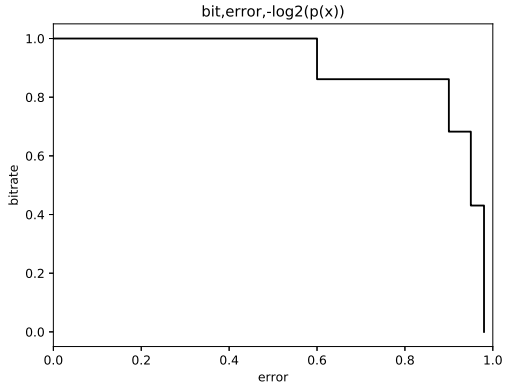

図 5　情報量を最大化する符号化を行った際のビットレートとエラーの関係

横軸がエラー，縦軸がビットレートである．実線で領域が 2 分されているが，これはそれぞれの符号化方法が達成可能なビットレートとエラーの領域の境界を示している．境界の左側は達成不可能領域であり，右側は達成可能領域，実線はその境界線である．そのため，誤り確率を最小化する符号化である図 4 においてエラーを 0.4 より小さくしたいのであれば，ビットレートを約 0.4 より大きくする必要がある．また，ビットレートを 0.4 より小さくしたいのであれば，エラーは 0.4 よりも大きくならざるを得ないことになる．情報量を最大化する符号化である図 5 では，達成可能領域が誤り確率を最小化する符号化よりも狭まっている．このことから小さい情報源においてビットレートとエラーの関係では，誤り確率を最小化する符号化の方が効率の良い符号化であることが分かる．

次に，誤り確率を最小化する符号化と情報量を最大化する符号化のビットレートと情報量の関係を図 6，図 7 に示す．

横軸が情報量，縦軸がビットレートである．情報源から出力されるすべての情報源シンボル数を n 個とすると，横軸の情報量は，自己情報量 $-\log_2 P_X(x)$ の総和である $-\sum_{i=1}^{n} \log_2 P_X(x_i)$ からどのくらい送信側で情報源シンボルを破棄せず正しく受信側に伝わる情報量を計測するため，受信側に伝わる情報量から情報源シンボルが持つ自己情報量の総和を割った比を示している．受信側に届く情報源シンボルの個数を l 個とし，受信側にもたらされる情報量を I_r と表すと，受信側にどのくらい情報量が届いたかを示す比は，

$$\frac{I_r}{-\sum_{i=1}^{n} \log_2 P_X(x_i)} = \frac{-\sum_{i=1}^{l} \log_2 P_X(x_i)}{-\sum_{i=1}^{n} \log_2 P_X(x_i)} \tag{7}$$

で示される．そのため，すべての情報源シンボルを破棄せずに正しく受信側に伝わるとすると I_r の l の個数がすべての情報源シンボルの個数 n になるため，その比は 1 となり，すべての

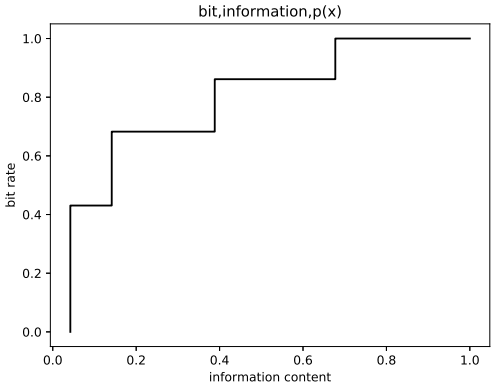

図 6　誤り確率を最小化する符号化を行った際のビットレートと情報量の関係

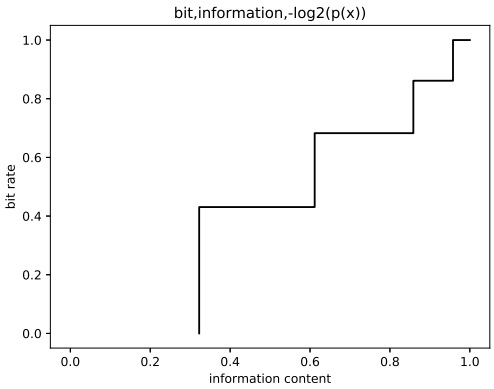

図 7　情報量を最大化する符号化を行った際のビットレートと情報量の関係

情報源シンボルを破棄すると $I_r = 0$ となり，比も 0 となる．

実線の左側は達成可能領域であり，右側は達成不可能領域である．そのため，例えば図 6 では，情報量を 0.5 より大きくしたいとするとビットレートは 0.8 よりも大きくなければならず，ビットレートを 0.5 よりも小さくしたいのであれば，情報量を 0.2 より小さくせざるをえないことが分かる．情報量を最大化する符号化である図 7 では，達成可能領域が誤り確率を最小化する符号化よりも大きくなっている．小さい情報源においてビットレートと情報量の関係では情報量を最大化する符号化の方がビットレートを固定したもとで多くの情報量を得られることがわかる．

最後に，誤り確率を最小化する符号化と情報量を最大化する符号化の情報量とエラーの関係

を図 8，図 9 に示す．

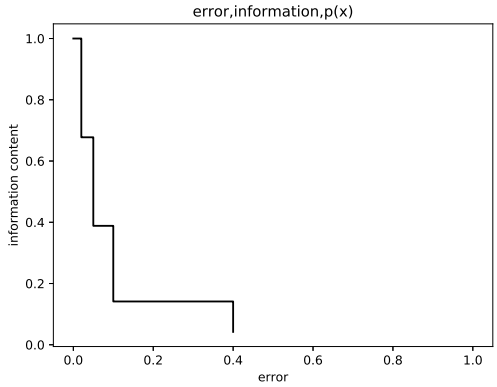

図 8　誤り確率を最小化する符号化を行った際の情報量とエラーの関係

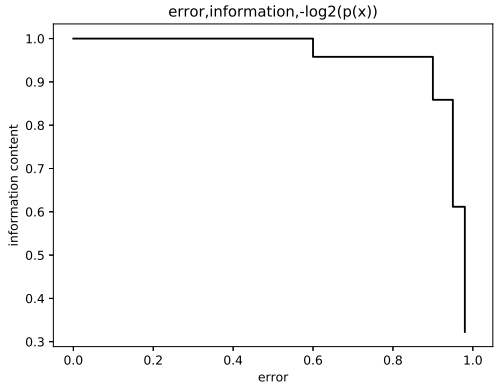

図 9　情報量を最大化する符号化を行った際の情報量とエラーの関係

横軸がエラー，縦軸が情報量である．実線の左側が達成可能領域であり，右側が達成不可能領域である．そのため，例えば図 8 において，情報量を全体の半分よりも大きくしたいとするとエラーは約 0.1 よりも小さくする必要がある．また，エラーを 0.4 より小さくしたいのであれば，情報量は高々約 0.1 取得できることが分かる．情報量を最大化する符号化である図 9 では，誤り確率を最小化する符号化に比べて達成可能領域が広くなっていることが分かる．このため，小さい情報源において情報量とエラーの関係ではエラーが許容していく際の振る舞いが異なることが分かり，情報量を最大化する符号化の方がエラーを許容していく際に得られる情

報量の減り方が緩やかであることが分かる．

4　検証と考察

本研究では，アルファベット拡大を利用し，できるだけ多くの情報源シンボルで 3 章同様のグラフを誤り確率を最小化する符号化方法と情報量を最大化する符号化方法で求め，その差異などを考察していく．また, その際に情報源シンボルの生起確率分布を変化させ，その挙動なども考察していく．

4.1　ビットレートとエラーの関係

アルファベット拡大を行う前の情報源シンボル生起確率を [0.9,0.1],[0.7,0.3],[0.5,0.5] としたときのビットレートとエラーの関係を示す．縦軸がビットレート，横軸がエラーを示している．図 10, 図 11 に生起確率 [0.9,0.1] としたもとでの誤り確率を最小化する符号化，情報量を最大化する符号化を示す．図 12，図 13 に生起確率 [0.7,0.3] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．図 14，図 15 には生起確率 [0.5,0.5] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．いずれもアルファベット拡大を行い，符号化の性能を計測した．

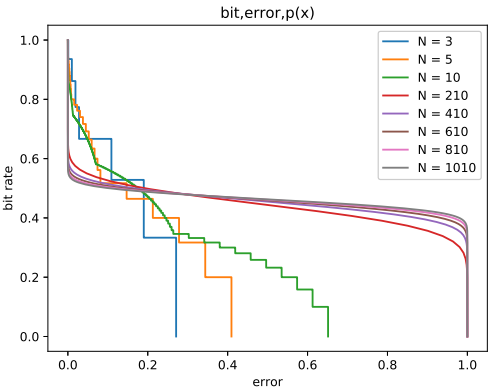

図 10　生起確率分布 [0.9,0.1] 時の誤り確率を最小化する符号化を行った際のビットレートとエラーの関係

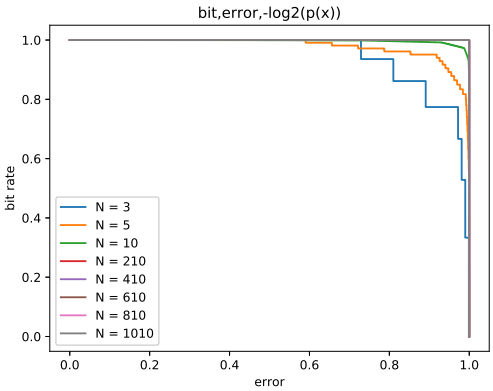

図 11　生起確率分布 [0.9,0.1] 時の情報量を最大化する符号化を行った際のビットレートとエラーの関係

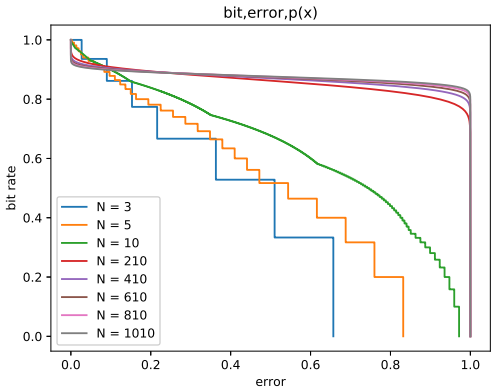

図 12　生起確率分布 [0.7,0.3] 時の誤り確率を最小化する符号化を行った際のビットレートとエラーの関係

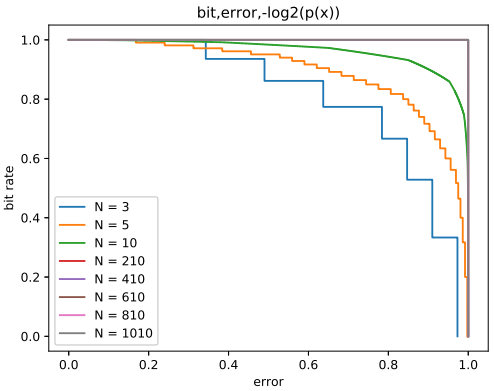

図 13　生起確率分布 [0.7,0.3] 時の情報量を最大化する符号化を行った際のビットレートとエラーの関係

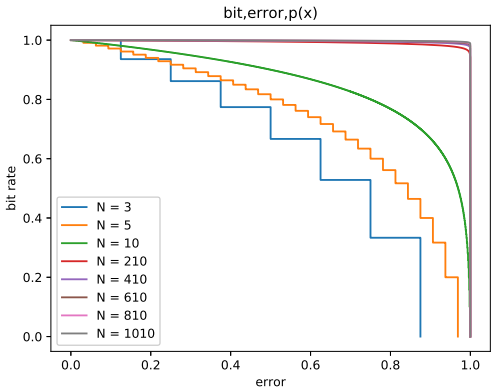

図 14　生起確率分布 [0.5,0.5] 時の誤り確率を最小化する符号化を行った際のビットレートとエラーの関係

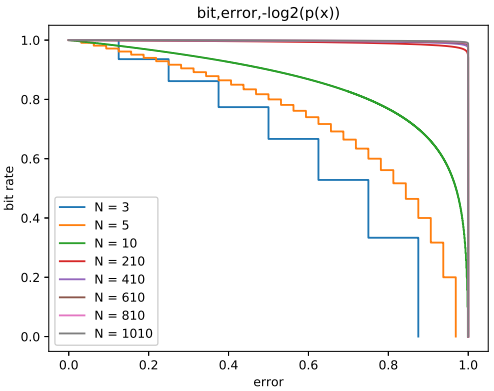

図 15　生起確率分布 [0.5,0.5] 時の情報量を最大化する符号化を行った際のビットレートとエラーの関係

図 10 において，アルファベット拡大を大きくすると達成可能な領域と達成不可能な領域を仕切る境界線がビットレート 0.5 付近で殆ど平らになっていることが分かる．理論的には $N \to \infty$ の極限でビットレート約 0.47 で完全に水平になることが知られている．つまり，エラーを少し許容したもとでビットレートを下げることができるため，誤り確率を最小化する符号化では，アルファベット拡大を大きくするとビットレートとエラーの関係においては効率が

良くなることがわかる．

一方，図 11 の情報量を最大化する符号化では，アルファベット拡大を大きくするとグラフ右上の達成可能領域が狭まることがわかる．例えば，N = 1010 においてビットレートを 1 よりも小さくしたいのであればエラーを殆ど許容しなければならず，エラーを小さくしたいのであれば，ビットレートは 1 にしなければならない．よって，情報量を最大化する符号化では，アルファベット拡大を大きくするとビットレートとエラーの関係において効率が悪くなることがわかる．

図 10，図 12，図 14 を比較すると，生起確率分布が等確率になるにつれてグラフ左上側の達成達成領域が小さくなっていることがわかる．

図 11，図 13，図 15 を比較すると，アルファベット拡大を十分に大きくするとどの生起確率分布でも差異が現れないが，アルファベット拡大が小さく，生起確率が等確率の時に達成可能領域が大きくなっていることがわかる．

図 14，図 15 では等確率に情報源シンボルが生起するので同じ関係性のグラフが得られる．

4.2 ビットレートと情報量の関係

アルファベット拡大を行う前の情報源シンボル生起確率を [0.9,0.1],[0.7,0.3],[0.5,0.5] としたときのビットレートと情報量の関係を示す．縦軸がビットレート，横軸が情報量を示している．図 16，図 17 に生起確率 [0.9,0.1] としたもとでの誤り確率を最小化する符号化，情報量を最大化する符号化を示す．図 18，図 19 に生起確率 [0.7,0.3] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．図 20，図 21 には生起確率 [0.5,0.5] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．

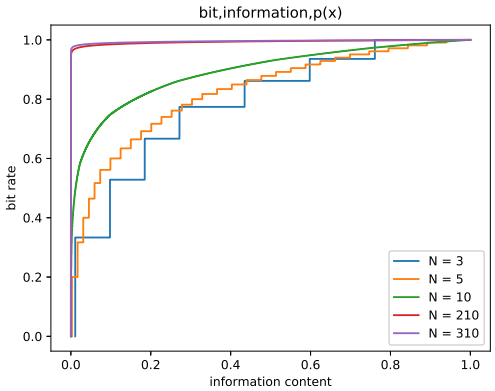

図 16　生起確率分布 [0.9,0.1] 時の誤り確率を最小化する符号化を行った際のビットレートと情報量の関係

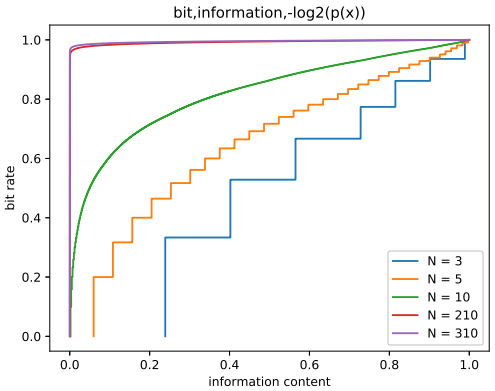

図 17　生起確率分布 [0.9,0.1] 時の情報量を最大化する符号化を行った際のビットレートと情報量の関係

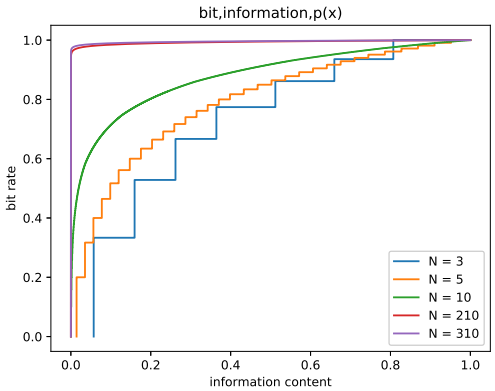

図 18　生起確率分布 [0.7,0.3] 時の誤り確率を最小化する符号化を行った際のビットレートと情報量の関係

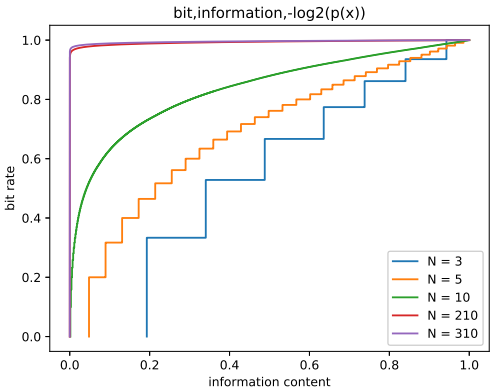

図 19　生起確率分布 [0.7,0.3] 時の情報量を最大化する符号化を行った際のビットレートと情報量の関係

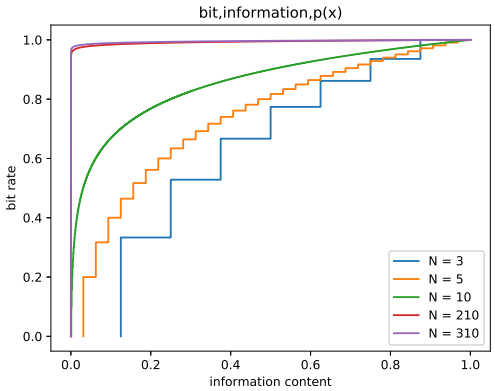

図 20　生起確率分布 [0.5,0.5] 時の誤り確率を最小化する符号化を行った際のビットレートと情報量の関係

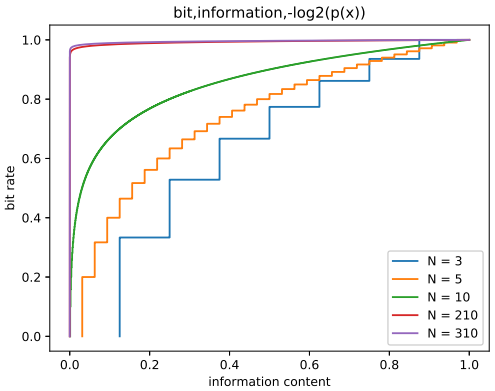

図 21　生起確率分布 [0.5,0.5] 時の情報量を最大化する符号化を行った際のビットレートと情報量の関係

図 17 においてグラフ左上の情報量とビットレートの関係の達成可能領域が図 16 よりも広いことが分かる．

図 16，図 18，図 20 を比較すると，誤り確率を最小化する符号化では，情報源シンボルの生起確率が偏るほど達成可能領域が狭くなる．一方，図 17，図 19，図 21 を比較すると情報量を最大化する符号化では，達成可能領域がシンボルの生起確率が偏るほど達成可能領域が広くなる．

ビットレートと情報量の関係においては，情報量を最大化する符号化を行ったほうが効率よく情報量を受信側に伝えることができる．

4.3　情報量とエラーの関係

アルファベット拡大を行う前の情報源シンボル生起確率を [0.9,0.1],[0.7,0.3],[0.5,0.5] としたときの情報量とエラーの関係を示す．縦軸が情報量，横軸がエラーを示している．図 22，図 23 に生起確率 [0.9,0.1] としたもとでの誤り確率を最小化する符号化，情報量を最大化する符号化を示す．図 24，図 25 に生起確率 [0.7,0.3] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．図 26，図 27 には生起確率 [0.5,0.5] としたもとでの誤り確率を最小化する符号化と情報源を最大化する符号化を示す．

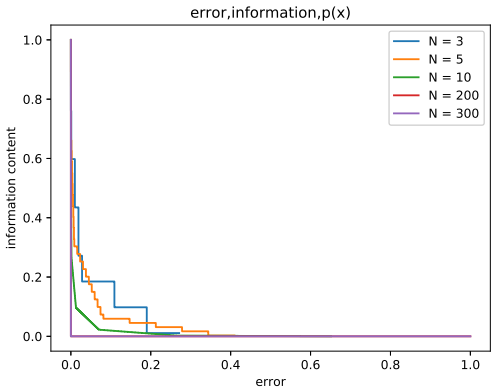

図 22　生起確率分布 [0.9,0.1] 時の誤り確率を最小化する符号化を行った際の情報量とエラーの関係

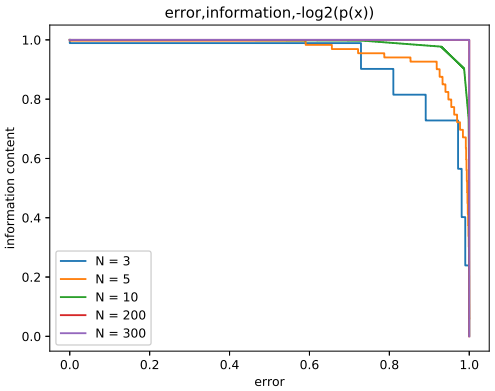

図 23　生起確率分布 [0.9,0.1] 時の情報量を最大化する符号化を行った際の情報量とエラーの関係

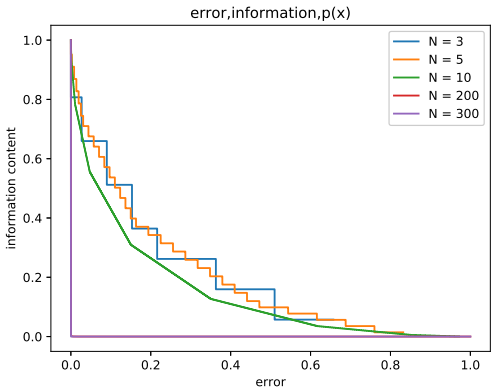

図 24　生起確率分布 [0.7,0.3] 時の誤り確率を最小化する符号化を行った際の情報量とエラーの関係

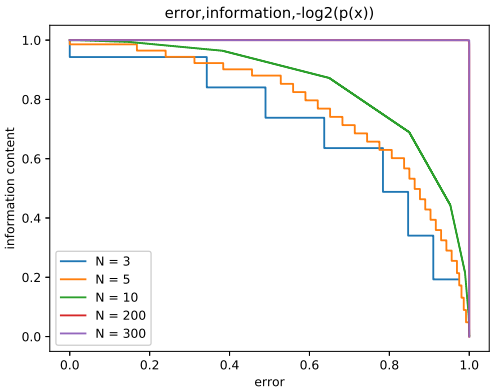

図 25　生起確率分布 [0.7,0.3] 時の情報量を最大化する符号化を行った際の情報量とエラーの関係

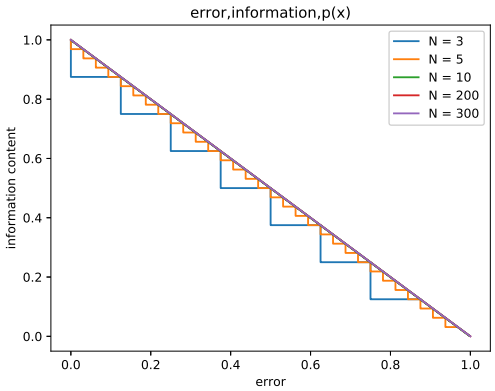

図 26　生起確率分布 [0.5,0.5] 時の誤り確率を最小化する符号化を行った際の情報量とエラーの関係

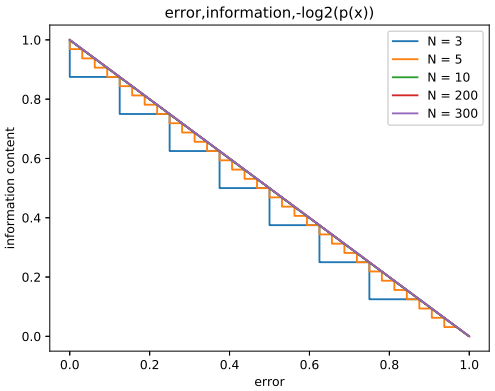

図 27　生起確率分布 [0.5,0.5] 時の情報量を最大化する符号化を行った際の情報量とエラーの関係

図 22 ではグラフ左下側の情報量とエラーの関係においての達成可能領域がアルファベット拡大を大きくしていくと狭くなる．一方，図 23 ではグラフ右上側の達成可能領域が広くなっている．

情報量とエラーの関係において誤り確率を最大化する符号化では，情報量を多く得るためにエラーを殆ど 0 にする必要がある．情報量を最大化する符号化では，エラーが大きくても情報量を多く得ることができる．

図 22，図 24，図 26 を比較する．図 22 や図 24 において，例えば情報量を 0.5 取得したいのであればエラーはほぼ 0 にする必要があり殆どの情報源シンボルを符号化しなければならいが，図 26 においてはエラーは 0.5 に許容することができる．誤り確率を最小化する符号化では，初期の情報源シンボルの生起確率が等確率である方が，生起確率に偏りがある情報源シンボルよりも情報量とエラーの関係において効率が良いことが分かる．

図 23，図 25，図 27 を比較する．図 23 や図 25 において，例えば情報量を 0.5 取得したいのであればエラーは殆ど許容することが出来き，符号化する情報源シンボルの数を少なくできるが，図 27 においてはエラーを殆ど許容することはできない．情報量を最大化する符号化では，初期の情報源シンボルの生起確率に偏りがある方が，生起確率が等確率であるよりも情報量とエラーの関係において効率が良いことが分かる．

4.4　エントロピーを最大化する符号化

此処までは生起確率や情報量に注目してきたが，此処からは情報量と生起確率の積であるエントロピー $H(X)$ に注目する．

拡大シンボル x^n を考える．ビットレートを下げたいが受信側に届くエントロピーをできるだけ大きくするようにしたい．符号語を持つ拡大シンボルが受信側にもたらすエントロピーは，$-P_{X^n}(x^n)\log_2 P_{X^n}(x^n)$ である．符号語を持たず，破棄された拡大シンボルの持つエントロピーは情報量が 0 のため 0 となる．受信側に届くエントロピーとは，符号語を持つ拡大シンボルのエントロピーの総和となるため符号語を持つ拡大シンボルを k 個とすると，$-\sum_{i=1}^{k} P_{X^n}(x^n{}_i)\log_2 P_{X^n}(x^n{}_i)$ となる．エントロピーを最大化する符号化を行うということは，拡大シンボルが持つエントロピーが大きいほうから符号化していく必要がある．

この情報源シンボルの持つエントロピーの大きいほうから符号語を割り当て，余ったシンボルを破棄する符号化方法を「エントロピーを最大化する符号化」と呼ぶことにする．

この符号化を行った時の符号化の性能を以下に示していく．情報源シンボル生起確率を [0.9,0.1] としたときのビットレートとエラーの関係を図 28 にビットレートと情報量の関係を図 29 に, 情報量とエラーの関係を図 30 に示す．

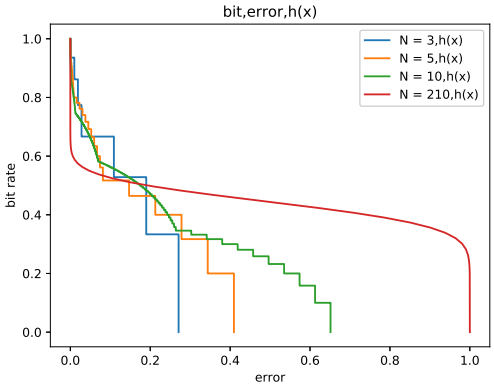

図 28　エントロピーを最大化する符号化を行った際のビットレートとエラーの関係

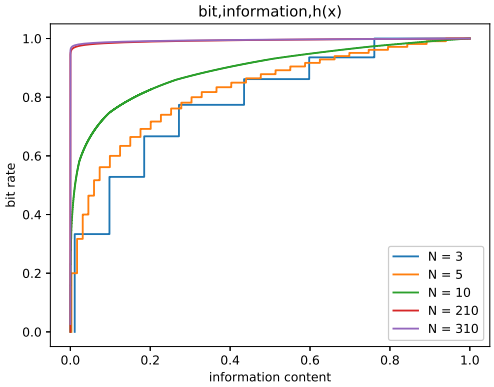

図 29　エントロピーを最大化する符号化を行った際のビットレートと情報量の関係

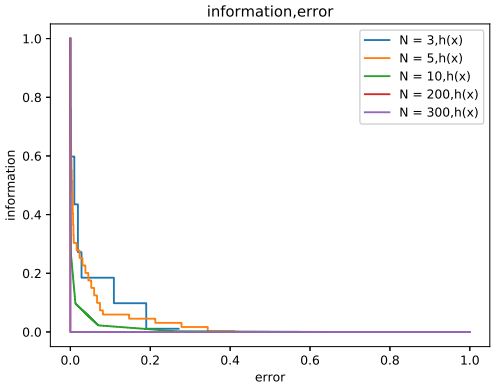

図 30　エントロピーを最大化する符号化を行った際の情報量とエラーの関係

どの関係性においても誤り確率を最小化する符号化と同じ計測結果が得られた．

しかし，アルファベット拡大が小さいとき誤り確率を最小化する符号化とは異なる計測結果が得られた．そこで 2 次拡大，3 次拡大の時のビットレートとエラーの関係を図 31, 図 32 に示す．誤り確率を最小化する符号化とエントロピーを最大化する符号化を比較する．青色が誤り確率を最小化する符号化，オレンジ色がエントロピーを最大化する符号化を示している．

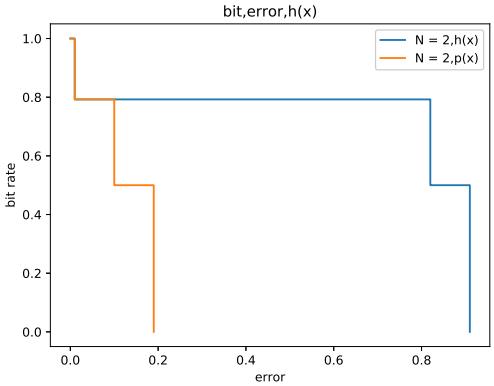

図 31　2 次拡大時のビットレートとエラーの関係比較

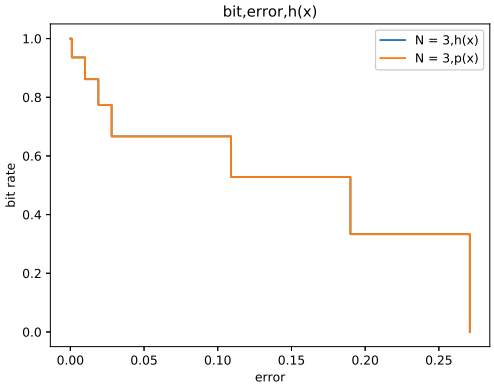

図 32　3 次拡大時のビットレートとエラーの関係比較

3 次拡大では，誤り確率を最小化する符号化とエントロピーを最大化する符号化では同じ計測結果が得られるが，2 次拡大では差異が現れている. この現象は，図 33 のグラフを用いて説明することができる．縦軸がエントロピー，横軸が生起確率を示している．

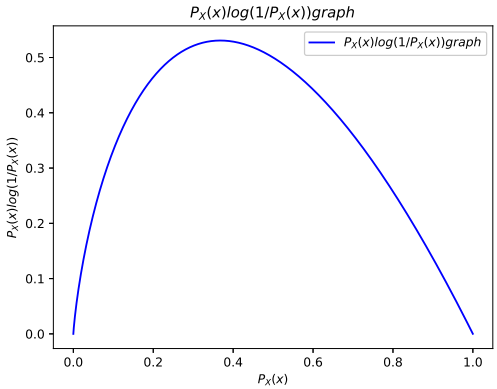

図 33　生起確率 $P_X(x)$, エントロピーのグラフ

このグラフの頂点は，微分をして傾きを 0 とすることで得られる．詳しい導出方法は付録 B に記す．

グラフの頂点は，(0.368,0.531) となる．つまり, 確率 0.368 のとき最大のエントロピーを取得する．アルファベット拡大を大きくしていくと情報源シンボルの個数は増えていき 1 つの情報源シンボルが持つ確率は小さくなる. すると, 殆どの確率が 0.368 よりも小さくなり，グラフの左側の単調増加部分でのみで議論ができる．単調増加部分では，確率が大きいものほど持つエントロピーが大きくなる．そのためアルファベット拡大を大きくし, 誤り確率を最小化する符号化を行うとエントロピーも最大化するように符号化していることになる．

アルファベット拡大が小さいとき，グラフの右側にある単調減少部分に確率を持つ情報源シンボルが存在するため，誤り確率を最小化する符号化と同様の計測結果は得られない．2 次拡大やアルファベット拡大をしないときには，単調減少部分に確率を持つ情報源シンボルよりも単調増加部分に確率を持つ情報源シンボルの方がエントロピーを多く持っているため，異なる計測結果になる．

5　まとめと今後の課題

本研究では，情報量を最大化する符号化においてビットレートを固定化したもとでアルファベット拡大をしたときのエラーや届く情報量の振る舞いについて考えた．情報量を最大化する符号化を行った際のビットレートとエラーの関係においては，アルファベット拡大を行っても効率が良くならない，一方情報量とビットレートの関係においては，誤り確率を最小化する符号化よりもビットレートを下げた上で受信側に伝えられる情報量を増やすことができる．また，情報量とエラーの関係においては，エラーを多く許容しても情報量を多く取得することが

できる．

情報量を最大化する符号化は，アルファベット拡大を大きくしていくと情報量の尺度から見れば効率が良い符号化方法であることが言えるが，ビットレートとエラーの関係においては，効率が良いとは言えない．

アルファベット拡大を行い誤り確率を最小化する符号化をすると，エントロピーを最大化することと同じになることが分かった．

今後の課題としては，この議論を拡張し記憶のある情報源にも適用できるようにしていくことである．本研究で行った記憶のない情報源では，適用できる範囲が限られるため，マルコフ情報源という，情報源のシンボルの生起確率がその時点から m 個前の出力シンボルが分かったうえで決まる，つまり情報源シンボルの生起確率が m 個前の出力シンボルに依存する情報源を考えることで議論を拡張できればと考えている．また，本実験ではアルファベットサイズ 2 の情報源を用意しアルファベット拡大を用いて情報源シンボルの個数を増やすやり方を採用したが，その方法に依存しない情報源を用意した上で同様の研究を行いたいと考えている．

謝辞

本研究を行うにあたり，細かく指導してくださった西新幹彦准教授に感謝の意を表する．

参考文献

[1] David J.C. MacKay , *Information Theory, Inference and Learning Algorithms* , Cambridge University Press , 2003.

[2] 宮川洋，情報理論，コロナ社，1979.

付録 A　ソースコード

A.1　誤り確率を最小化する符号化においてビットレートとエラーの関係を出力するプログラム

```
#ビットレート、エラーの関係、情報量を最小化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
expand_p = []
a = []
b = []
list_N = [3,5,10,210,410,610,810,1010]

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for i in range(0,len(list_N),1):
    N = list_N[i]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))


    pb = [[0 for i in range(2)] for j in range(len(comb))]

    #確率大きいほうから符号化

    if(N >15):
      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
        if(j==0):
          a.append(1-pb[j][0])

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a = np.append(a,((a[j-1])**2-(pb[j][0]*comb[j])**2)/(a[j-1]+pb[j][0]*comb[j]))

        b.append((1/N)*math.log2(pb[j][1]))

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
          c.append(expand_p[i])

      for j in range(0,len(c),1):

        if(j==0):
            a.append(1-c[j])

        elif(j<len(c)):
```

```
            a.append(a[j-1] - c[j])

        b.append((1/N)*math.log2(j+1))

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

    plt.plot(a,b,label = "N = %i" %N)
    plt.legend(loc='best')

    a = []
    b = []
    expand_p = []
    comb = []

plt.xlabel("error")
plt.ylabel("bit rate")

plt.title("bit,error,p(x)")
plt.show()
```

A.2 情報量を最大化する符号化においてビットレートとエラーの関係を出力するプログラム

```
# bit,error の関係, 情報量を最大化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
expand_p = []
a = []
b = []
list_N = [3,5,10,210,410,610,810,1010]

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for i in range(0,len(list_N),1):
    N = list_N[i]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))


    pb = [[0 for i in range(2)] for j in range(len(comb))]

    expand_p.reverse()#情報量の大きいほうから符号化

    if(N >15):
      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
```

```
        if(j==0):
          a.append(1-pb[j][0])

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a = np.append(a,((a[j-1])**2-(pb[j][0]*comb[j])**2)/(a[j-1]+pb[j][0]*comb[j]))

        b.append((1/N)*math.log2(pb[j][1]))

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
          c.append(expand_p[i])

      for j in range(0,len(c),1):

        if(j==0):
            a.append(1-c[j])

        elif(j<len(c)):
            a.append(a[j-1] - c[j])

        b.append((1/N)*math.log2(j+1))

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

    plt.plot(a,b,label = "N = %i" %N)
    plt.legend(loc='best')

    a = []
    b = []
    expand_p = []
    comb = []

plt.xlabel("error")
plt.ylabel("bit rate")
plt.title("bit,error,-log2(p(x))")
plt.show()
```

A.3 誤り確率を最小化する符号化においてビットレートと情報量の関係を出力するプログラム

```
#bit rate,information の関係　誤り確率最小化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
```

```
expand_p = []
a = []
b = []
list_N = [3,5,10,210,310]

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for i in range(0,len(list_N),1):
    N = list_N[i]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))


    pb = [[0 for i in range(2)] for j in range(len(comb))]

    #確率大きいほうから符号化

    if(N >15):
      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
        if(j==0):
          a.append(-1*math.log2(pb[j][0]))#情報量

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a = np.append(a,(a[j-1] + -1*math.log2(pb[j][0])*comb[j])) #情報量

        b.append((1/N)*math.log2(pb[j][1]))#ビットレート

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
          c.append(expand_p[i])

      for j in range(0,len(c),1):

        if(j==0):
            a.append(-1*math.log2(c[j]))#情報量

        elif(j<len(c)):
            a.append(a[j-1] + (-1*math.log2(c[j]))) #情報量

        b.append((1/N)*math.log2(j+1))

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

    for i in range(0,len(a),1):
        a[i] = a[i]/a[-1]

    plt.plot(a,b,label="N = %i" %N)
    plt.legend(loc='best')
```

```
    a = []
    b = []
    expand_p = []
    comb = []

plt.xlabel("information content")
plt.ylabel("bit rate")
plt.title("bit,information,p(x)")

plt.show()
```

A.4 情報量を最大化する符号化においてビットレートと情報量の関係を出力するプログラム

```
#bit rate,information の関係 情報量を最大化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
expand_p = []
a = []
b = []
list_N = [3,5,10,210,310]

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for i in range(0,len(list_N),1):
    N = list_N[i]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))

    expand_p.reverse() #情報量の大きいほうから符号化
    pb = [[0 for i in range(2)] for j in range(len(comb))]

    if(N >15):
      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
        if(j==0):
          a.append(-1*math.log2(pb[j][0]))#情報量

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a = np.append(a,(a[j-1] + -1*math.log2(pb[j][0])*comb[j])) #情報量

        b.append((1/N)*math.log2(pb[j][1]))#ビットレート

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
```

```
          c.append(expand_p[i])

      for j in range(0,len(c),1):

        if(j==0):
            a.append(-1*math.log2(c[j]))#情報量

        elif(j<len(c)):
            a.append(a[j-1] + (-1*math.log2(c[j]))) #情報量

        b.append((1/N)*math.log2(j+1))

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

    for i in range(0,len(a),1):
        a[i] = a[i]/a[-1]

    plt.plot(a,b,label="N = %i" %N)
    plt.legend(loc='best')

    a = []
    b = []
    expand_p = []
    comb = []

plt.xlabel("information content")
plt.ylabel("bit rate")
plt.title("bit,information,-log2(p(x))")

plt.show()
```

A.5 誤り確率を最小化する符号化において情報量とエラーの関係を出力するプログラム

```
#情報量とエラーの関係 誤り確率を最大化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
expand_p = []
list_N = [3,5,10,200,300]
a = []
b = []

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for j in range(0,len(list_N),1):
    N = list_N[j]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))
```

```
    #P(x) 確率の大きいほうから符号化した場合
    pb = [[0 for i in range(2)] for j in range(len(comb))]


    if(N >= 16):

      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
        if(j==0):
          a.append(1-pb[j][0]) #エラー
          b.append(math.log2(1/pb[j][0])) #情報量

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a.append(((a[j-1])**2-(pb[j][0]*comb[j])**2)/(a[j-1]+pb[j][0]*comb[j]))
          b.append(b[j-1]+math.log2(1/pb[j][0])*comb[j])

      for i in range(0,len(b),1):
          b[i] = b[i]/b[-1]

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
          c.append(expand_p[i])

      for j in range(0,len(c),1):

        if(j==0):
            a.append(-1*math.log2(c[j]))#情報量
            b.append(1-c[j])#エラー

        elif(j<len(c)):
            a.append(a[j-1] + (-1*math.log2(c[j]))) #情報量
            b.append(b[j-1] - c[j])#エラー

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

      for i in range(0,len(a),1):
        a[i] = a[i]/a[-1]

    plt.plot(b,a,label = "N = %i" %N)
    a = []
    b = []
    comb = []
    expand_p = []


plt.legend(loc='best')
plt.xlabel("error")
plt.ylabel("information content")
plt.title("error,information,p(x)")

plt.show()
```

A.6 情報量を最大化する符号化において情報量とエラーの関係を出力するプログラム

```
#情報量とエラーの関係 情報量を最大化する符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
expand_p = []
list_N = [3,5,10,200,300]
a = []
b = []

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for j in range(0,len(list_N),1):
    N = list_N[j]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))

    #P(x) 確率の大きいほうから符号化した場合
    pb = [[0 for i in range(2)] for j in range(len(comb))]

    expand_p.reverse()


    if(N >= 16):

      for j in range(0,N+1,1):
        pb[j][0] = expand_p[j]
        pb[j][1] = comb[j]
        if(j==0):
          a.append(1-pb[j][0]) #エラー
          b.append(math.log2(1/pb[j][0])) #情報量

        elif(j<N+1):
          pb[j][1] = pb[j][1]+pb[j-1][1]
          a.append(((a[j-1])**2-(pb[j][0]*comb[j])**2)/(a[j-1]+pb[j][0]*comb[j]))
          b.append(b[j-1]+math.log2(1/pb[j][0])*comb[j])

      for i in range(0,len(b),1):
          b[i] = b[i]/b[-1]

    else:
      c = []
      for i in range(0,N+1,1):

        if(comb[i] >= 2):
          for j in range(1,comb[i]+1,1):
            c.append(expand_p[i])

        else:
          c.append(expand_p[i])

      for j in range(0,len(c),1):
```

```
        if(j==0):
            a.append(-1*math.log2(c[j]))#情報量
            b.append(1-c[j])#エラー

        elif(j<len(c)):
            a.append(a[j-1] + (-1*math.log2(c[j]))) #情報量
            b.append(b[j-1] - c[j])#エラー

      for k in range(0,len(a)-1,1):
        if(k==0):
          a.insert(k,a[k])
          b.insert(k+1,b[k+1])
        else:
          n = k*2
          a.insert(n,a[n])
          b.insert(n+1,b[n+1])

      for i in range(0,len(a),1):
        a[i] = a[i]/a[-1]

    plt.plot(b,a,label = "N = %i" %N)
    a = []
    b = []
    comb = []
    expand_p = []


plt.legend(loc='best')
plt.xlabel("error")
plt.ylabel("information content")
plt.title("error,information,-log2(p(x))")

plt.show()
```

A.7 エントロピーを最大化する符号化においてビットレートとエラーの関係を出力するプログラム

```
#ビットレート、エラーの関係 エントロピー最大化をする符号化
import math
import numpy as np
import matplotlib.pyplot as plt

p = [0.9,0.1]
pb= []
comb = []
h = []
expand_p = []
a = []
b = []
list_N = [3,5,10,210]

def comb_count(n, r):
    return math.factorial(n) // (math.factorial(n - r) * math.factorial(r))

for i in range(0,len(list_N),1):
    N = list_N[i]

    for i in range(0,N+1,1):
        expand_p.append(2**(N*math.log2(p[0]) - i*math.log2(p[0]) + i*math.log2(p[1])))
        comb.append(comb_count(N,i))


    pb = [[0 for i in range(3)] for j in range(len(comb))]
```

```
for i in range(0,len(expand_p),1):
    h.append(expand_p[i]*math.log2(1/expand_p[i]))

#エントロピーの大きいほうから符号化

if(N >15):
  for j in range(0,N+1,1):
    pb[j][0] = h[j]
    pb[j][1] = expand_p[j]
    pb[j][2] = comb[j]

  pb.sort()
  pb.reverse()

  for j in range(0,N+1,1):
    if(j==0):
      a.append(1-pb[j][1])#エラー

    elif(j<N+1):
      pb[j][2] = pb[j][2]+pb[j-1][2]
      a = np.append(a,((a[j-1])**2-(pb[j][1]*comb[j])**2)/(a[j-1]+pb[j][1]*comb[j])) #エラー

    b.append((1/N)*math.log2(pb[j][2]))#ビットレート

else:
  for j in range(0,N+1,1):
    pb[j][0] = h[j]
    pb[j][1] = expand_p[j]
    pb[j][2] = comb[j]

  pb.sort()
  pb.reverse()

  c = []
  for i in range(0,len(pb),1):

    if(pb[i][2] >= 2):
      for j in range(0,pb[i][2],1):
        c.append(pb[i][1])

    else:
      c.append(pb[i][1])

  for j in range(0,len(c),1):

    if(j==0):
        a.append(1-c[j])#エラー

    elif(j<len(c)):
        a.append(a[j-1] - c[j]) #エラー

    b.append((1/N)*math.log2(j+1))

  for k in range(0,len(a)-1,1):
    if(k==0):
      a.insert(k,a[k])
      b.insert(k+1,b[k+1])
    else:
      n = k*2
      a.insert(n,a[n])
      b.insert(n+1,b[n+1])

plt.plot(a,b,label="N = %i,h(x)" %N)
plt.legend(loc='best')


a = []
b = []
```

```
    expand_p = []
    comb = []
    h = []

plt.xlabel("error")
plt.ylabel("bit rate")
plt.title("bit,error,h(x)")

plt.show()
```

付録 B　生起確率 $P_X(x)$ とエントロピーの関係グラフの頂点導出

グラフの頂点は，微分をして傾きを 0 とすることで得られる．縦軸 $P_X(x)\log_2 \frac{1}{P_X(x)}$ を y とし，横軸 $P_X(x)$ を x とする．

$$
\begin{aligned}
y &= x\log_2 \frac{1}{x} \\
\frac{dy}{dx} &= x'\log_2 \frac{1}{x} + x(\log_2 \frac{1}{x})' \\
&= \log_2 \frac{1}{x} - \frac{1}{\log_e 2}
\end{aligned}
$$

傾きに 0 を代入する．

$$
\begin{aligned}
\log_2 \frac{1}{x} - \frac{1}{\log_e 2} &= 0 \\
\log_2 \frac{1}{x} &= \frac{1}{\log_e 2} \\
\log_2 \frac{1}{x} &= \log_2 2^{\frac{1}{\log_e 2}} \\
\frac{1}{x} &= 2^{\frac{1}{\log_e 2}} \\
x &= \frac{1}{2^{\frac{1}{\log_e 2}}} \approx 0.368
\end{aligned}
$$

x を代入して y を求める．

$$
\begin{aligned}
y &= \frac{1}{2^{\frac{1}{\log_e 2}}} \log_2 2^{\frac{1}{\log_e 2}} \\
y &= \frac{1}{2^{\frac{1}{\log_e 2}} \log_e 2} \approx 0.531
\end{aligned}
$$

頂点 (0.368,0.531) が得られる．