第 0 章
序文

1995 年にインフルエンザ菌の完全ゲノム配列 1,830,135 文字が同定されて以来、ヒトを含む数多くの生物種の完全ゲノム配列がインターネットで公表されている。いままでブラックボックスであった生命の「設計図／プログラム」が、少なくとも機械語レベルでは参照できるようになったことになる。地球上すべての生物が同じ「プログラム言語」を使用し、DNA やたんぱく質などのハードウェアは全生物共通であることから、生物の本質はゲノム配列という「ソフトウェア」にあると言える。

これに伴い、ゲノム配列情報に基づく全く新しい生命科学が登場した。ラジオの原理や振る舞いを解析するとき（あるいは修理しようとするとき）、誰しもがまず設計図を参照するように、生物の原理や振る舞いを解析するとき（あるいは病気を治療するとき）、生物の設計図であるゲノム配列情報をいかに利用するかが重要な鍵となる。

今までに数多くのゲノム解析用ソフトウェアが開発された。しかしこれら既存のソフトウェアを利用するだけでは、新規の生物学的発見をすることは難しい。大量に蓄積されているゲノムデータから、人々が驚くような生物学知見を発見するためには、独自のソフトウェアを開発したり、複数のソフトウェアを組み合わせたりすることが不可欠である。そしてそのためには、生物学とコンピュータプログラミングの両方の知識を持った人材育成が必要である。

慶應義塾大学湘南藤沢キャンパスでは、1998 年からゲノム配列解析のためのプログラミング実習の授業を学部生および大学院生向けに実施してきた。1998 年〜2000 年は「情報処理２ B」、2001 年〜現在までは「ゲノム解析プログラミング」という科目名で、毎年数十人の学生が履修している。

本書は 7 年間に渡る実習授業で実際に使用している教材に手を加えて教科書にしたものである。この授業の履修者の中には、その後ゲノム解析の研究を行い、在学中に国際ジャーナルに論文が掲載された学生も少なくない。本書を手にした読者においても、近い将来プログラミングを駆使してさまざまな生物学的知見を発見されることを願ってやまない。

冨田　勝

本書はバイオインフォマティクスの中でも特に核酸・アミノ酸配列解析法および遺伝子発現データの解析法を学習したい学部生、大学院生、研究者ならびに技術者向けに書かれた実習用のテキストです。バイオインフォマティクスという学問が注目を集めるようになってから 10 年以上が経とうとしていますが、新規実験技術を用いて次々と生産・公開される多量の生化学データを効果的に解析するための手法を模索すべく、この学問は現在でも発展を続けています。一方で、配列のアラインメントや遺伝子発現データのクラスタリングなど理論の整備が進み、確立されつつある分野もあります。本書ではそのような分野の中から、バイオインフォマティクスの中で重要な位置を占めているものを取り上げました。

本書は互いにオーバーラップを含む統計解析、アルゴリズム、多変量解析の 3 つの部分から構成されています。統計解析としてまず、基本的な統計学や情報理論を用いたシグナル配列の解析法を解説しました。次にバイオインフォマティクスの分野で重要なアルゴリズムである、アラインメント、隠れマルコフモデル、RNA の二次構造予測を取り上げました。これらは再帰アルゴリズムを使って簡潔に記述されています。再帰アルゴリズムに馴染みが薄い方のために付録に詳しい説明を載せましたので、まずはこちらを見て勉強して下さい。最後に多変量解析の例として、遺伝子発現データの解析とコドンバイアスの解析を載せました。

解説にはなるべく図を使い、無味乾燥になりがちな理論をなるべく分かりやすく説明するように心がけました。アルゴリズムを理解する上で最も良い方法は実際にプログラムを組んでみることだと長年の経験から考えています。本書ではアルゴリズムを良く理解できるように、理論的背景の解説と並行してプログラムへの実装法も示しました。これは他書にはあまり見られない、本書の最大の特徴でもあります。プログラムは C 言語で書かれていますので、馴染みのない方には C 言語初心者用のテキストを用意してあります。適宜数理科学の WEB ページよりダウンロードして勉強することをお勧めします。

本書は慶應義塾大学湘南藤沢キャンパス（SFC）の授業「ゲノム解析プログラミング実習」および「バイオインフォマティクスアルゴリズム実習」で実際に使われており、高い効果を挙げています。本書を手に取った読者の方々がバイオインフォマティクスの理論を学び、演習問題を通じて実際にプログラムを組み、その楽しさを味わって頂けるなら、著者としてこれほど嬉しいことはありません。また本書の中で不適切な記述などを見つけられた方は著者までご一報頂ければ幸いです。

なおこのテキストを執筆するにあたって、同じ研究室の鈴木治夫氏にはバクテリアのコドンバイアスのデータを提供して頂いた上で様々なご意見を頂き、杉本昌弘氏および谷内江望氏には本文中で扱う数式の計算を助けて頂きました。また藤森茂雄氏には階層的クラスタリングを行うプログラムを提供して頂きました。東京大学大学院新領域創成科学研究科の有田正規助教授には原稿を読んで頂き、本テキストをより良い作品に仕上げるための様々なコメントを頂きました。これらの方々に厚く御礼申し上げます。

最後に私の長期に渡る執筆活動を辛抱強く最後まで支えて頂いた数理科学編集部の平勢耕介氏に感謝の意を表したいと思います。

斎藤輪太郎

目　次

第 1 章
シグナル配列の統計解析

DNA や RNA 上には転写開始を指示する配列、翻訳開始を指示する配列、ゲノムの組み換えを指示する配列など様々なシグナル配列が存在します。またタンパク質配列（アミノ酸配列）にも細胞内局在シグナル配列など多種多様なシグナルが存在します。そこで我々が知りたいのは、生体内で使われるシグナル配列はどのような配列パターンを持っているかということと、そのシグナル配列の機能は何か、ということです。

本章では主に前者の問題に焦点を当てて、シグナル配列の存在を検出する方法を情報理論および統計解析のアプローチから考えてゆきます。

1.1 シグナル配列

ゲノムにはどのような情報が書いてあるのでしょうか？一番最初に思い浮かぶのは、遺伝子情報でしょう。遺伝子情報は合成される RNA やタンパク質の情報が書いてある部分です。生命現象の多くが RNA やタンパク質によって成り立っていることを考えると、この領域は極めて重要であると言えます。

しかしながら、ゲノムの全領域が遺伝子をコードしているわけではありません。特に真核生物では、遺伝子をコードしている部分はほんの数パーセントと言われています。つまりゲノムには遺伝子になる部分とならない部分があるのです。では生体内では遺伝子をコードしている部分はどのように見分けられているのでしょうか？

実はゲノムには遺伝子情報だけでなく、遺伝子の発現を調整するための情報も書かれているのです。図 1.1 を見て下さい。DNA 上にコードされている遺伝子を含む領域はまず、mRNA という分子に写し取られます。これを転写と呼びます。次に、mRNA の情報をもとにタンパク質が合成されます。これを翻訳と呼びます。

注意すべき点は、DNA の中で転写される領域は一部であり、また mRNA

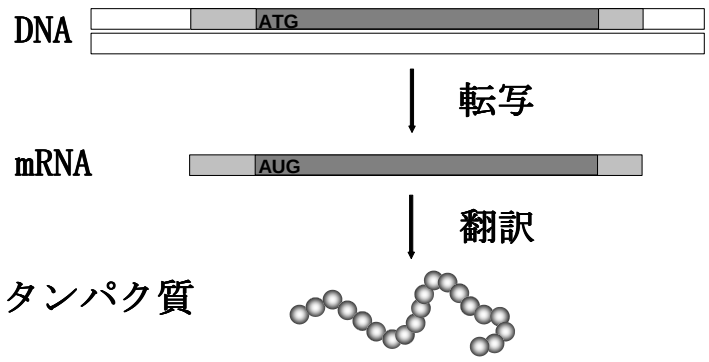


図 1.1　遺伝子の発現

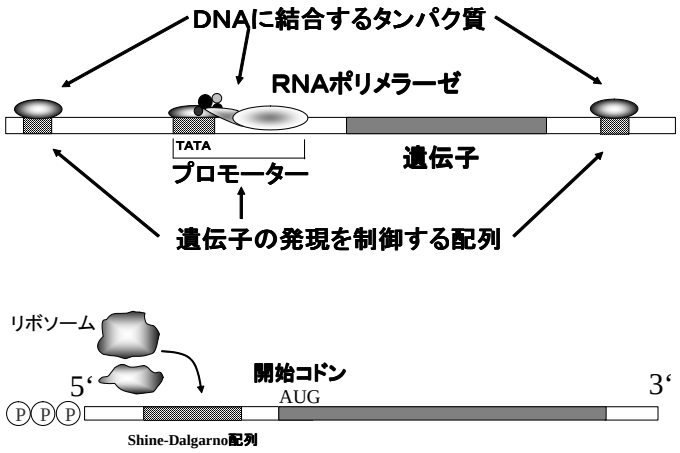


図 1.2　遺伝子の境界領域のシグナル配列

の中でも翻訳されるのは一部である、ということです。すると生体の中では、DNA の中で mRNA になる領域と、mRNA の中で翻訳される領域が見分けられていることになります。

図 1.2 のように転写が開始されるときは、転写因子と呼ばれるタンパク質が DNA の転写開始領域の近くに結合します。ある転写因子が結合する部位の塩基配列には一定の傾向があります。例えば、大腸菌の転写開始部位から 10 塩基上流には TATAAT という配列が頻繁に観測されます [1]。また翻訳を開始するときは、リボソームと呼ばれる細胞内小器官が開始コドン付近に結合します。バクテリアの場合、リボソームが結合する部位には Shine-Dalgarno(SD) 配列と呼ばれる A,G(プリン）に富んだ配列が観測されます [2]。図 1.3 に示すように、リボソーム中の 16S rRNA の 3’ 末端が Shine-Dalgarno 配列と対合するのです [3]。従って、SD 配列は 16S rRNA の 3’ 末端の配列と相補的になっています。

このようにゲノム上には転写の開始位置を示したり、翻訳開始部位を示したりと何か生化学反応を指定するような配列パターンが数多く存在します。これを**シグナル配列**[*1)] と呼ぶことにしましょう。生物が使用するシグナル配列には様々なものがあります。例えば mRNA 前駆体のエクソン・イントロン境界の切断部位を指定するシグナル配列はほとんど GT-AG という配列です。このようにほぼ一定のパターンを持つシグナル配列もありますが、中には曖昧なシグナルもあります。例えば表 1.1 は大腸菌のいくつかの遺伝子の開始コドン周辺

---

*1）　シグナル配列とともによく使われる用語として「コンセンサス配列」がある。これは複数の配列間で保存されている配列パターンである。

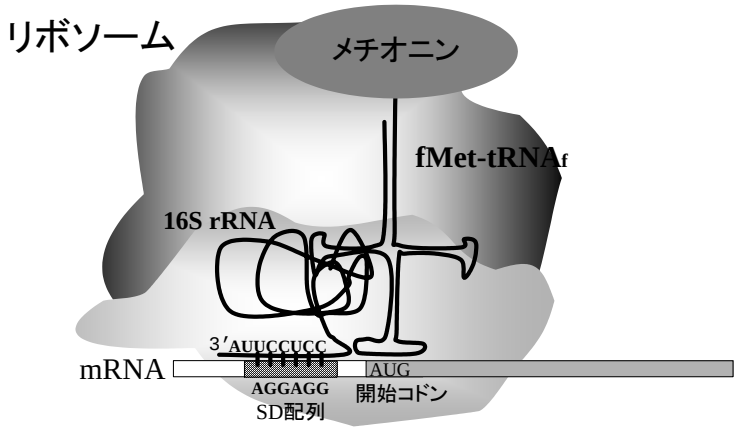


図 1.3　原核細胞における翻訳開始

リボソーム中の 16S rRNA の 3’ 末端が mRNA の開始コドン上流の SD 配列と対合して翻訳が始まる。従って、16S rRNA の 3’ 末端の配列と SD 配列は相補的になっている。

上流配列	開始コドン	コード領域	機能注釈
ttacagagtacacaacatcc	atg	aaacgcatta	[thr operon leader peptide:thrL]
aaggtaacgaggtaacaacc	atg	cgagtgttga	[aspartokinase I]
atggaagttaggagtctgac	atg	gttaaagttt	[homoserine kinase:thrB]
cacgagtactggaaaactaa	atg	aaactctaca	[threonine synthase:thrC]
aatgataaaaggagtaacct	gtg	aaaaagatgc	[hypothetical protein:b0005]
atttcctgcaaggactggat	atg	ctgattctta	[hypothetical protein:yaaA]
gtttaaagagaaatactatc	atg	acggacaaat	[transaldolase B:talB]

表 1.1　大腸菌開始コドン周辺の塩基配列の例

	−27	−26	−25	−24	−23	−22	−21	−20	−19	−18	−17	−16
A	27.6	28.9	28.4	27.2	28.5	30.4	27.9	31.0	31.4	29.0	32.3	33.3
T	28.3	26.7	29.8	29.4	27.1	28.7	32.3	27.5	30.0	30.8	27.6	28.9
C	22.9	23.0	22.8	23.7	23.2	22.2	21.9	21.2	21.6	21.5	20.1	20.5
G	21.2	21.5	18.9	19.7	21.2	18.7	18.0	20.3	17.0	18.7	19.9	17.3

	−15	−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4
A	32.4	34.1	35.5	33.0	36.1	34.0	27.7	34.8	36.6	31.5	33.7	35.5
T	28.6	24.0	24.2	21.9	13.5	14.8	14.5	14.6	20.4	26.2	25.1	24.9
C	19.6	19.2	18.4	16.0	14.7	10.5	9.8	11.5	13.3	16.7	19.0	21.6
G	19.4	22.8	21.9	29.1	35.7	40.7	48.0	39.1	29.7	25.6	22.1	18.0

	−3	−2	−1	**0**	**1**	**2**	3	4	5	6	7	8
A	38.4	26.6	29.3	83.0	0.0	0.0	46.4	34.1	30.2	40.4	38.5	29.4
T	19.4	30.8	29.6	3.1	100.0	0.0	14.4	21.6	31.0	16.5	27.1	28.0
C	18.5	25.6	25.0	0.0	0.0	0.0	18.0	27.3	19.2	20.2	19.9	19.9
G	23.8	17.0	16.1	13.9	0.0	100.0	21.2	16.9	19.6	22.9	14.4	22.7

図 1.4　大腸菌開始コドン周辺の塩基の分布
表の上段に開始コドンの最初の文字からの距離を示した。開始コドンの位置を灰色にし、開始コドンの上流は負の値にした。

の塩基配列を列挙したものです。開始コドン上流には SD 配列が存在するはずですが、一見して共通のパターンは見当たりません。

そこで、各塩基 ACGT の割合がどのようになっているかを大腸菌の約 4000 個の遺伝子について調べたのが表 1.4 です。大腸菌の場合開始コドンの多くは AUG なので、開始コドンの位置 (+0〜+2) の塩基は A、T、G に偏っているのが分かります。そして上流の塩基を見ると、-11〜-8 にかけて塩基が A,G(プリン）に偏っているのが観測されます。これが SD 配列です。このように単に配列を並べただけでは観測が難しかった傾向が、各位置ごとの塩基の分布を計算することにより浮き彫りになったのが分かると思います。

1.2　エントロピー計算によるシグナル配列の保存性の定量

表 1.4 の上流の配列を見ると、例えば-16 の位置も A が 33%もあり、偏って

いるように見えます。-16 の位置と、-11〜-8 の位置の塩基の偏りはどちらが強いのでしょうか？それを議論するためには、偏りの程度を定量化する指標が必要になってきます。その指標の 1 つとして考えられるのが、**エントロピー**です [5]。エントロピー H は以下のように定義されます。

$$H = -\sum_{i=\mathrm{a,c,g,t}} P_i \log_2 P_i$$

ここで P_i はある位置における塩基 i の頻度を表し、$\sum_{i=\mathrm{a,c,g,t}} P_i = 1, 0 \leq P_i \leq 1$ です。$\log_2 0$ は定義されませんが、ロピタルの定理 [6] より、

$$\lim_{p\to 0} p \log_2 p = \lim_{p\to 0} \frac{(\log_2 \frac{1}{p^{-1}})'}{(p^{-1})'} = \lim_{p\to 0} -\frac{p}{\log 2} = 0$$

となるので、$0 \log 0 = 0$ と定義します。

エントロピー H は 0 に近ければ近いほど、塩基が大きく偏っていることを意味し、2 に近ければ近いほど塩基が均等に出現していることを意味します。例えばある位置において、A が 100%を占め、残りが 0 の場合、

$$H = -1 \log_2 1 - 0 \log_2 0 - 0 \log_2 0 - 0 \log_2 0 = 0$$

となります。A,C,G,T が 25%ずつ均等に存在した場合は、

$$\begin{aligned} H &= -0.25 \log_2 0.25 - 0.25 \log_2 0.25 - 0.25 \log_2 0.25 - 0.25 \log_2 0.25 \\ &= -0.25 \times -2 \times 4 = 2 \end{aligned}$$

です。その間の偏りの例として、A が 50%, C,G が 25%, T が 0%の場合は、

$$\begin{aligned} H &= -0.5 \log_2 0.5 - 0.25 \log_2 0.25 - 0.25 \log_2 0.25 - 0 \log_2 0 \\ &= 0.5 + 0.5 + 0.5 + 0 = 1.5 \end{aligned}$$

となり、先ほどの 2 例の間の数となります。

ここで情報理論で扱う情報量の概念についてごく簡単に触れておきます。私たちは普段テレビや新聞、本などから様々な情報を得ています。各地の天気の情報は毎日報じられていますが、「山形県鶴岡市で 4 月 7 日の日中の気温は 12 ℃でした」という情報と、「山形県鶴岡市で 4 月 7 日の日中の気温は 40 ℃でした」という情報ではどちらが情報量が多いと感じるでしょうか？多くの人は後者の方ではないかと思います。何故なら前者はほとんど当たり前のように起こる出来事なので、大した情報のようには感じませんが、後者は起こる確率が極めて低いため、ものすごい情報だと感じるからです。情報理論では、このように確率が低い事象が起こったときの通報を情報量が多いと考え、情報の生起確率を P としたときに、その情報量を $-\log P$ で表します。

配列解析の話に戻ると、ゲノム上のある領域において、塩基 A が観測される

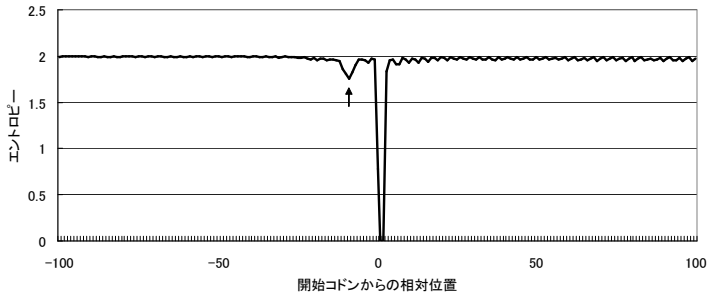


図 1.5　大腸菌開始コドン周辺のエントロピー

確率が $\frac{1}{2}$ で、実際に A が観測された場合、得た情報量は $-\log_2 \frac{1}{2} = 1$ ビットとなります[*2)]。エントロピーとは、ゲノム上のある位置において得られる情報量の期待値[*3)]のことを指すのです。

さてエントロピーを使って実際に大腸菌の開始コドン周辺のエントロピーを表 1.4 をもとに計算したのが図 1.5 です。大部分の領域は定位置にシグナル配列を含んでいないと考えられ、塩基の分布も均等になっているので、エントロピーは多くの位置では 2 に近い値を示しています。しかし、開始コドンの 1 文字目の多くは A であり、2 文字目、3 文字目は T、G なので、開始コドンの場所のエントロピーは 0 に近くなっています。そして開始コドンの上流の-9 の位置のエントロピーは 1.75 であり、ちょうど逆向きのピークになっています。この周辺に SD 配列が存在することが分かります。

$x > 0$ に対し $\log_e x \leq x - 1$ から、$\log_e \frac{1}{4P_i} \leq \frac{1}{4P_i} - 1$ が成立します。従って、

$$-\sum_{i=\mathrm{a,c,g,t}} P_i \log_e P_i + \sum_{i=\mathrm{a,c,g,t}} P_i \log_e \frac{1}{4} = \sum_{i=\mathrm{a,c,g,t}} P_i \log_e \frac{1}{4P_i}$$
$$\leq \sum_{i=\mathrm{a,c,g,t}} P_i(\frac{1}{4P_i} - 1) = 0$$

より、

$$-\sum_{i=\mathrm{a,c,g,t}} P_i \log_e P_i \leq \log_e 4$$

が成立します。両辺を $\log_e 2$ で割ると、

$$-\sum_{i=\mathrm{a,c,g,t}} P_i \log_2 P_i \leq 2$$

---

*2)　対数の底を 2 にする場合、単位はビットになる。

*3)　期待値の定義は次の通り。確率変数 X が $p_1, p_2, \cdots, p_n$ の確率で、値 $x_1, x_2, \cdots x_n$ を取るとき $(P(X = x_i) = p_i)$、期待値 $E(X) = \sum_{i=1}^{n} p_i x_i$ となる。但し $\sum_{i=1}^{n} p_i = 1$ である。

となります。

また $P_i \geq 0$ より、$-P_i \log_2 P_i \geq 0$ なので、$\sum_{i=\mathrm{a,c,g,t}} -P_i \log_2 P_i \geq 0$ が成立します。結局、

$$0 \leq H \leq 2$$

が成立することになります。但し $H = 0$ となるのは、ある塩基 i について $P_i = 1$ であり、その他の全ての塩基 $j(i \neq j)$ について $P_j = 0$ のときに限ります。また $H = 2$ となるのは全ての塩基 i について、$P_i = \frac{1}{4}$ が成り立つときに限ります。

H はコンセンサスが強ければ 0 に近づき、コンセンサスが弱ければ 2 に近づく指標ですが、$H^* = 2 - H$ は逆にコンセンサスが弱ければ 0 に近づき、強ければ 2 に近づく指標となります。しかし $H^* \geq 0$ となるので、コンセンサスが全くない（つまり 4 つの塩基が完全に 1/4 の確率で出現する）H^* 場合でも、H^* の期待値 $E(H^*)$ は 0 にはなりません。解析対象となる塩基配列の数 N が十分大きければ、

$$E\left(H^* - \frac{3}{2N}\log_e 2\right) = 0$$

となるので (B.3 参照)、コンセンサスが全くない場合の指標の期待値を 0 にしたい場合、$H^* - \frac{3}{2N}\log_e 2$ を使うといいでしょう [7]。

1.3 増加情報量

エントロピーは塩基の均等使用からの偏りを測る指標でした。しかし生物種によっては A,C,G,T の 4 塩基が均等使用から大きくずれているものもあります。例えば、マイコプラズマ菌の一種である *M. genitalium* は AT 含有量が高いゲノムをもっています。このような生物種では、仮にある位置の塩基が A と T に偏っていたとしても、直ちにその場所に何らかのシグナルがあるとは言えません。このようなケースでは、その場所が A と T に偏っているのは、単にゲノム全体の塩基組成の傾向を反映しているだけと考えたほうが妥当です。そこでゲノム全体の塩基組成と比べて注目している位置にどれだけ強いコンセンサスがあるかを測る指標を考えましょう。

今、あるゲノム配列の塩基 i の組成を B_i とします。ここで、$\sum_i B_i = 1$ です。次に注目している位置の塩基 i の組成を P_i とします。ここでも $\sum_i P_i = 1$ です。ある塩基 i が観測されたとき、ゲノム全体の塩基組成を考えたときに得られる情報量は、$-\log_2 B_i$ です。しかし特定の位置に着目した場合は $-\log_2 P_i$ であり、その差は

$$-\log_2 B_i - -\log_2 P_i = \log_2 P_i - \log_2 B_i = \log_2 \frac{P_i}{B_i} \tag{1.1}$$

です。この式が特定の位置の塩基組成／ゲノム全体の塩基組成の対数をとった

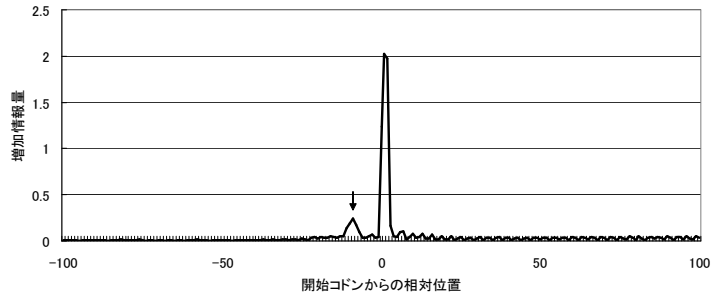


図 1.6　大腸菌開始コドン周辺の増加情報量

ものであることが分かります。特定の位置に塩基 i が観測されたときの式 1.1 の期待値は、塩基 i の出現頻度 P_i を掛けて

$$\sum_i P_i(\log_2 P_i - \log_2 B_i) = \sum_i P_i \log_2 \frac{P_i}{B_i} \tag{1.2}$$

となります。この式 1.2 が増加情報量 [8] と呼ばれる指標になります。増加情報量は、ゲノム全体の塩基組成 B_i に比べて、注目している位置の塩基組成 P_i がどれだけ異なるかということを表しています。ゲノム全体の塩基組成と注目している位置の塩基組成が全く同じ場合、増加情報量は 0 になることが直ちに導かれます。また注目している位置の塩基組成がゲノム全体の塩基組成から離れれば、それだけ増加情報量は高い値を示します。ゲノム全体の塩基組成が均等のとき、つまり全ての塩基 i について、$B_i = 1/4$ のときは増加情報量は H^* に一致するので、本質的にはエントロピーと同じ指標になります。

$x > 0$ に対し $\log_e x \leq x - 1$ から、

$$-\log_2 \frac{B_i}{P_i} \geq -\frac{1}{\log_e 2}(\frac{B_i}{P_i} - 1)$$

が成立し、両辺に P_i を掛けて総和をとると、

$$\sum_i P_i \log_2 \frac{P_i}{B_i} \geq= -\log_2 \sum_i B_i \frac{P_i}{B_i} = -\log_2 1 = 0 \tag{1.3}$$

となるので、増加情報慮は常に 0 以上の値になることが分かります。なお 0 になるのは、$P_i = B_i, i = \mathrm{a, c, g, t}$ のときに限ります。

図 1.6 に大腸菌の開始コドン周辺の増加情報量を示します。まず、開始コドンの位置に大きなピークが見られることが分かります。そしてその上流に SD 配列のピーク (矢印で示した部分) が見られます。

1.4　塩基の偏りの統計的有意性

これまではエントロピーや増加情報量という値を用いてコンセンサスの強さ

を定量化する方法を紹介してきました。しかし統計的に考えて、本当にシグナル配列があるかどうかを測る指標も同時に考える必要があります。つまり例えばある位置で増加情報量が増えるのはそこに有意な塩基の偏りがあるためなのか、あるいはただ偶然の変動によるものなのかをしっかりと考える必要があるのです。仮に塩基の偏りが有意であれば、それはシグナル配列の存在を示唆する 1 つの結果になり得ます。

統計的有意性を考える上で、最も頻繁に使われるのが、χ^2 値です。今帰無仮説 H_0 を、ある位置の塩基の偏り P_i はゲノム全体の偏り B_i と差がないという命題とします。

$$H_0 : B_i = P_i \quad \text{for all } i$$

次に対立仮説 H_1 を、ある位置の塩基の偏りはゲノム全体の偏りと差があるという命題とします。

$$H_1 : B_i \neq P_i \quad \text{for some } i$$

この場合、χ^2 値は以下の式で定義されます。

$$\chi^2\text{値} = N \sum_{i=\mathrm{a,c,g,t}} \frac{(P_i - B_i)^2}{B_i} \tag{1.4}$$

但し N は解析対象の配列数です。仮説 H_0 が正しいとき、χ^2 値はどのような振る舞いを示すのでしょうか？ゲノム全体の偏りと差が全くない多数の位置に対して、1.4 の式に基づいて χ^2 値を算出してゆき、ヒストグラムを作ると理論的には図 1.7 に示すようなものになるはずです。

図 1.7 に示されるような分布を自由度 3 の χ^2 分布[*4] と呼びます (χ^2 分布の概要については、B.2 参照)。式 1.4 の値がどのような分布を示すかを集計したとき、理論的に図 1.7 の形になるとき、式 1.4 は自由度 3 の χ^2 分布に従うといい、以下のように表現します。

$$N \sum_{i=\mathrm{a,c,g,t}} \frac{(P_i - B_i)^2}{B_i} \sim \text{自由度 3 の} \chi^2\text{分布}$$

図 1.7 の分布を調べると、χ^2 値が 12.84 を超える可能性は 0.5%未満となることが分かります。従って、ある 1 つの位置に着目したとき、χ^2 値が 12.84 を超える値だったなら、それが偶然に起こる確率は極めて低いため、H_0 が成立するとは考えにくいことになります[*5]。つまりその位置にはゲノム全体の傾向

---

*4) ここで自由度と言っているのは、自由に動ける変数の数のこと。この例では $P_\mathrm{a}, P_\mathrm{c}, P_\mathrm{g}, P_\mathrm{t}$ がそれに該当するが、$P_\mathrm{a} + P_\mathrm{c} + P_\mathrm{g} + P_\mathrm{t} = 1$ という制約が付いているため、3 つが決まると残りの 1 つは自動的に決まってしまう。従って自由度は 3 となる。

*5) あくまで 1 つの位置に着目した場合。例えば 1,000,000 箇所の位置について χ^2 値を調べれば、偶然高い値を示すようなものが現れる可能性は十分にあるため、その中の 1 つが 12.84 を超えていたとしても、有意ということはできない。

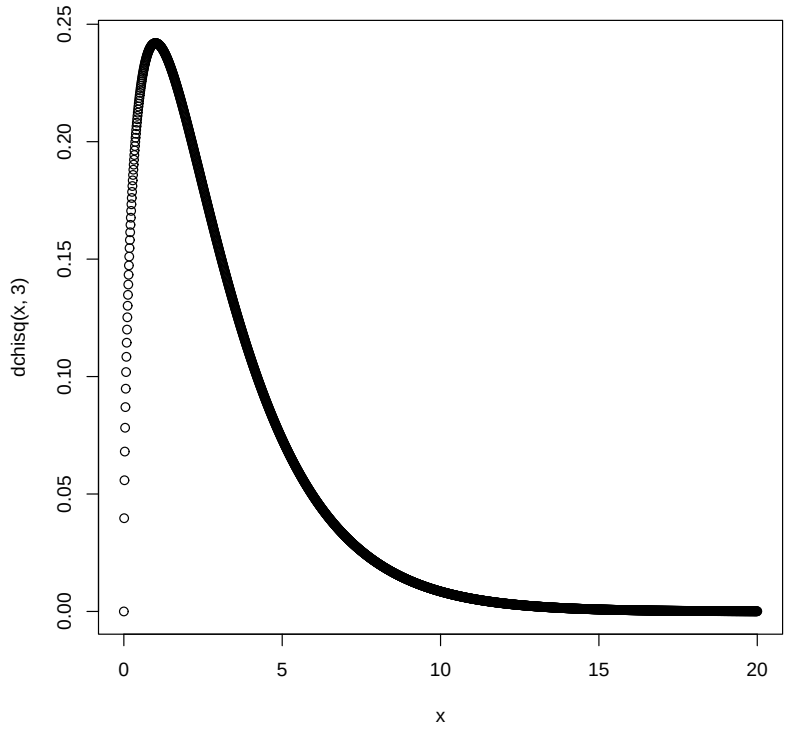


図 1.7　自由度 3 の χ^2 分布

横軸が χ^2 値、縦軸がその値を取る頻度を表す。なお、本図は R 言語を用いて作成した。

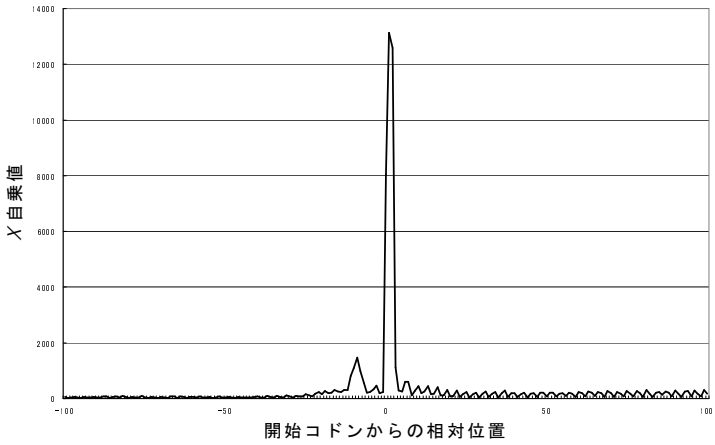


図 1.8　大腸菌開始コドン周辺の塩基の出現頻度の χ^2 値

とは異なる塩基の偏りがあると考えていいでしょう。もっと正確な言い方をすれば、仮説 H_0 は 0.5%の有意水準で棄却されるのです。

実際に大腸菌の開始コドン周辺の χ^2 を調べたのが、図 1.8 です。開始コドンのところに大きなピークが見られ、その上流手前に SD 配列による塩基の有意な偏りがあることが分かります。

有意性の検証は以下に示す対数尤度比統計量 [8] と呼ばれる値を使っても行うことができます。

$$\text{対数尤度比統計量} = 2N \sum_i P_i \log_2 \frac{P_i}{B_i} \tag{1.5}$$

解析対象となる位置の数 N が大きくなると、対数尤度比統計量は χ^2 値に近づくことが知られています (B.4 参照)。式 1.5 を見ると、対数尤度比統計量は増加情報量と解析対象となる位置の数の積になっています。この式より塩基の偏りの程度が一定でも、位置の数が増えるとそれだけ対数尤度比統計量は大きくなることが分かります。言い換えると、解析対象のデータの数が増せば、統計的な信頼性が上がってくるため、対数尤度比統計量や χ^2 値は有意な値（つまり大きな値) を示すということです。

1.5　配列パターンの出現頻度解析

これまでは１塩基あるいは１つの位置ごとに塩基の頻度や偏りを評価していました。しかし、１塩基のみがシグナル配列として働くことはどちらかというと稀で、通常は数塩基から数十塩基がシグナル配列として働きます。従って、

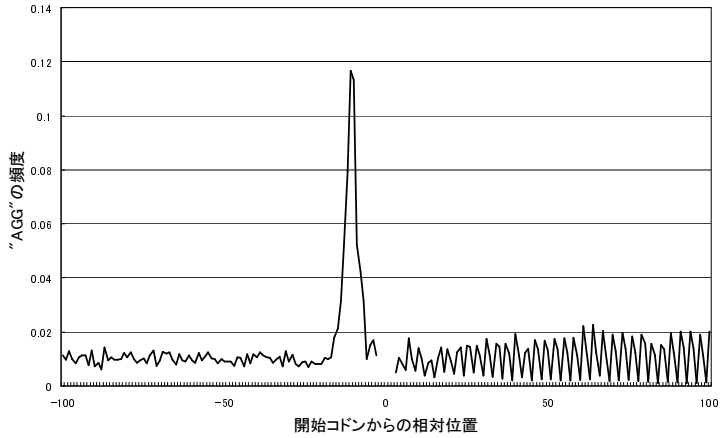


図 1.9　大腸菌開始コドン周辺の”AGG”の出現頻度
開始コドンの位置の頻度は省略した。

これまで述べてきた方法だけでは、どのような塩基配列パターンがシグナル配列として働いているのかを調べるには不十分です。そこで複数の塩基からなるシグナル配列を検出する簡単な方法について考えましょう。

仮に複数の配列の中に共通のシグナル配列があるとすれば、そのシグナル配列の配列パターンが多量に観測されることが考えられます。さらにシグナル配列の出現位置がほぼ決まっていれば、ある位置に集中してそのシグナル配列の配列パターンがたくさん観測されることが考えられます。

ある位置におけるある配列パターンの出現頻度は以下のように表すことができます。

$$\text{出現頻度} = \frac{n}{N}$$

但し、n はある位置におけるあるパターンの観測数、N は解析対象となった配列の数です。

図 1.9 は大腸菌の開始コドン周辺における”AGG”の出現頻度をプロットしたものです。開始コドン直前の -11 の位置に”AGG”のピークがあることが分かります。”AGG”は SD 配列の一部であり、この結果から SD 配列の存在が示唆されます。

この結果からどの程度”AGG”が出現するのかが分かりましたが、次にその出現頻度がどれだけ偶然からは程遠いものかを評価する方法について考えましょう。今、シグナル配列が含まれていない領域で特定の位置において特定の配列パターンが観測される確率は p であるとしましょう。N 本の配列の中で、それ

ぞれの特定の位置でその配列パターンが観測された本数 X が n 本となる確率 $P(X = n)$ は、

$$P(X = n) = \frac{N!}{n!(N-n)!} p^n (1-p)^{N-n} \tag{1.6}$$

となります。従って、n 本以上の配列が観測される確率は、

$$P(X \geq n) = \sum_{i=n}^{N} \frac{N!}{i!(N-i)!} p^i (1-p)^{N-i} \tag{1.7}$$

となります。Np および $N(1-p)$ が十分大きいとき[*6]、X は正規分布と呼ばれる分布に従うことが知られています。ここで Z スコア (Z Score) と呼ばれる指標を以下の式によって定義しましょう。

$$\text{Z スコア} = \frac{n - Np}{\sqrt{Np(1-p)}} \tag{1.8}$$

Z スコアは標準正規分布と呼ばれる分布に従うことが知られており (B.1 参照)、以下のように表されます。

$$\text{Z スコア} \sim N(0, 1)$$

Z スコアは平均が 0、標準偏差が 1 です。実際に大腸菌の開始コドン周辺の”AGG”の出現頻度をプロットした結果を図 1.10 に示します。開始コドンの直前の位置に SD 配列の存在を示す、統計的に有意なピークが観測されていることが分かります。またそれ以外の位置では Z スコアはさほど有意な値を示していない (0 の周辺を振動している) ことが分かります。

実際にはどの塩基配列パターンが有意なのかは分かりません。そこでどのパターンが統計的に有意に出現するかを調べるためには、3 塩基なら $4^3 = 64$ 通り、4 塩基なら $4^4 = 256$ 通りの全ての塩基パターンについて網羅的に Z スコアを調べる必要があります。

表 1.2 に 4 種類のバクテリアについて、開始コドンの上流と下流において顕著に出現する 4 塩基のパターンを Z スコアが高い順に 1 位から 5 位までを示しました。大腸菌では、”agga”というパターンが最も顕著に出現していることが分かります。これは SD 配列の一部で、掲載した 4 種の 16S rRNA の 3’ 末端には”agga”と相補的な配列があることが分かります。”agga”というパターンは、枯草菌、メタノコッカス菌においても確認できます。5 位になってしまいますが、シアノバクテリアでも”agga”というパターンが観測されることが分かりました。なお、シアノバクテリアの開始コドン直前には”acc”というパターンが顕著に観測されることが発見されており [4]、それが一位として検出できています。

---

*6) $Np \geq 5$ かつ、$N(1-p) \geq 5$ が 1 つの基準になるだろう。

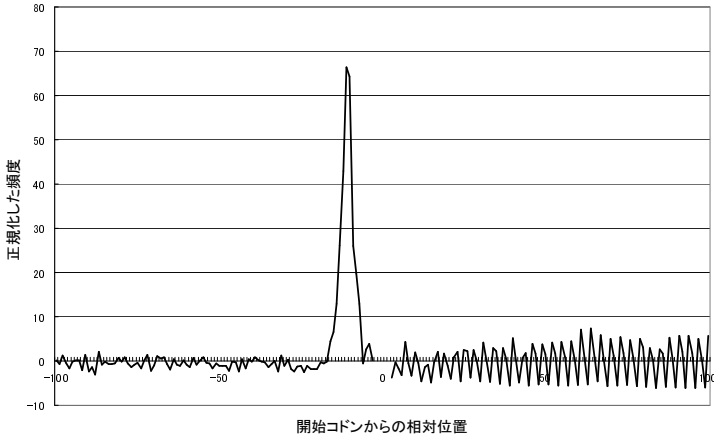


図 1.10　大腸菌開始コドン周辺の”AGG”の出現頻度の Z Score

Z スコアを算出するときの p の値は、開始コドン上流 500 塩基〜下流 500 塩基の中に含まれる”AGG”の割合で推定している。

Escherichia coli

16S rRNA 3- terminal: gcggttggatcacctcctta3 ccttaaagaa

Expected SD Sequence: ttctttaagg 5taaggaggtgatccaaccgc

Pat.	Z-Sc.	Pos.	Pat.	Z-Sc	Pos.
agga	94.97	-11	aaaa	29.46	3
ggag	82.94	-10	aata	19.28	3
aagg	58.15	-11	agta	18.28	3
gagg	53.08	-11	ctaa	16.76	4
gaga	42.23	-9	attg	15.29	15

Bacillus subtilis

16S rRNA 3- terminal: ggctggatcacctcctttct3 aaggatattt

Expected SD Sequence: aaatatcctt 5agaaaggaggtgatccagcc

Pat.	Z-Sc.	Pos.	Pat.	Z-Sc	Pos.
ggag	132.85	-12	aaaa	27.03	3
gagg	111.09	-12	aata	18.52	3
agga	95.56	-13	taaa	16.46	5
gggg	84.60	-12	ctaa	14.54	4
aagg	82.83	-14	aaac	14.31	6

Methanococcus jannaschii

16S rRNA 3- terminal: actgcggctggatcacctcc3 tgagaaaaaa

Expected SD Sequence: ttttttctca 5ggaggtgatccagccgcagt

Pat.	Z-Sc.	Pos.	Pat.	Z-Sc	Pos.
ggtg	189.61	-9	gtga	29.17	3
gtga	172.77	-8	tatg	20.94	8
aggt	67.18	-10	atgg	19.99	9
tggt	53.57	-10	atga	18.60	9
tgat	46.59	-7	aatg	12.63	8

Synechocystis PCC6803

16S rRNA 3- terminal: gtggctggatcacctccttt3 aagggagacc

Expected SD Sequence: ggtctccctt 5aaaggaggtgatccagccac

Pat.	Z-Sc.	Pos.	Pat.	Z-Sc	Pos.
aacc	18.94	-4	ctaa	15.51	4
taac	17.52	-5	acta	13.40	3
cacc	16.49	-4	ttcc	12.18	8
aaac	16.45	-5	ctga	12.10	4
agga	15.49	-13	acca	11.51	3

表 1.2　4 種のバクテリアの開始コドン周辺に出現する有意な 4 塩基パターン

Pat. は配列パターン、Z-Sc. は Z スコア、Pos. は開始コドンからの距離を表す。同時に各生物の 16S rRNA の 3' 末端の配列およびそれと相補的な配列を示した。

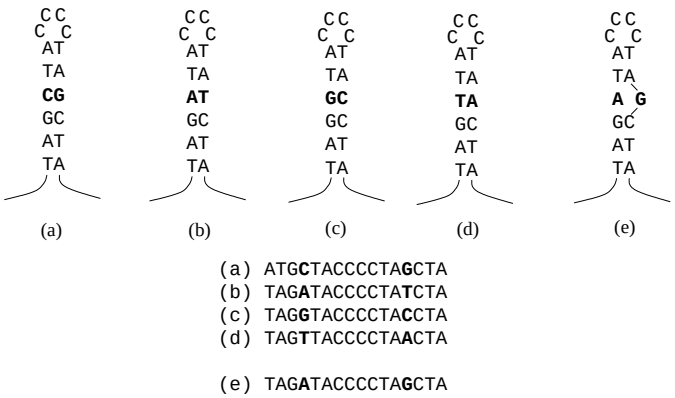


図 1.11　RNA 二次構造上の塩基間の相互作用

1.6　塩基間の相互作用の解析

これまで説明してきた事柄は主にコンセンサス配列、つまり塩基の保存性に関するものでした。確かに生体内である一定の働きをするシグナル配列は一定の配列パターンを持っていることが多いといえます。しかしシグナルとなり得るのは配列パターンそのものだけではありません。例えば tRNA はどの塩基とどの塩基が結合しているかという二次構造も重要です。この場合、塩基配列にはどのような特徴が見られるのでしょうか？

図 1.11(a)〜(d) に同一の二次構造を持つ RNA 配列を示しました。この中で太文字になっている塩基を変えていますが、ステム上、それと反対側にある塩基を常に対合するようにしています。つまり A に対して T、C に対して G が反対側に来るようにしています。すると太文字の位置は常に結合するため、結果的に同じ二次構造が維持されます。この場合は配列の保存性よりもむしろ塩基間の相互作用が重要になっています。塩基間の相互作用とは簡単に言えば、ある位置の塩基と他の位置の塩基の出現に何らかの従属関係があることだと言えます。

では、塩基間の相互作用の強さを測るにはどのような指標が考えられるでしょうか？ここで位置 u および v におけるエントロピーをそれぞれ $H_{\mathrm{u}}, H_{\mathrm{v}}$ とします。次に位置 u,v における塩基の結合エントロピー $H_{\mathrm{u,v}}$ を以下のように定義します。

$$H_{\mathrm{u,v}} = -\sum_{i=\mathrm{a,c,g,t}} \sum_{j=\mathrm{a,c,g,t}} P_{\mathrm{u}:i,\mathrm{v}:j} \log_2 P_{\mathrm{u}:i,\mathrm{v}:j} \tag{1.9}$$

但し $P_{\mathrm{u}:i,\mathrm{v}:j}$ は、位置 u に塩基 i が観測され、かつ位置 v に塩基 j が観測される配列の割合です。結合エントロピーは位置 u の塩基と、位置 v の塩基を一続きのパターンと見立てたときのそのパターンのエントロピーと考えることがで

u	v
A	A
A	C
A	G
A	T
C	A
C	C
C	G
C	T
G	A
G	C
G	G
G	T
T	A
T	C
T	G
T	T

(a)

u	v
A	C
A	C
A	G
A	G
C	G
C	G
C	T
C	T
G	T
G	T
G	A
G	A
T	A
T	A
T	C
T	C

(b)

u	v
A	C
A	C
A	C
A	C
C	G
C	G
C	G
C	G
G	T
G	T
G	T
G	T
T	A
T	A
T	A
T	A

(c)

図 1.12　相互情報量の計算例

きます。今位置 u の塩基 i の頻度を $P_{\mathrm{u}:i}$、位置 v の塩基 j の塩基の頻度を $P_{\mathrm{v}:j}$ とします。位置 u の塩基と位置 v の塩基の間に何の従属関係もなければ、十分多くの配列に対し、

$$P_{\mathrm{u}:i,\mathrm{v}:j} = P_{\mathrm{u}:i} P_{\mathrm{v}:j} \tag{1.10}$$

が成立するはずです。この場合、

$$H_{\mathrm{u,v}} = H_{\mathrm{u}} + H_{\mathrm{v}} \tag{1.11}$$

が成立します。また逆に位置 u と v の位置の塩基に完全な従属関係が成り立つ場合、すなわち位置 u の塩基が決まれば位置 v の塩基が完全に決まる場合は、

$$H_{\mathrm{u,v}} = H_{\mathrm{u}} = H_{\mathrm{v}} \tag{1.12}$$

が成立します。

ここで、位置 u と位置 v の間の塩基の相互作用の強さを表す相互情報量 $I_{\mathrm{u,v}}$ は以下のように定義されます [9]。

$$I_{\mathrm{u,v}} = H_{\mathrm{u}} + H_{\mathrm{v}} - H_{\mathrm{u,v}} \tag{1.13}$$

位置 u の塩基と位置 v の塩基の間に何の従属関係もなければ、式 1.11 より $I_{\mathrm{u,v}} = 0$ であり、また逆に完全な従属関係が成り立つ場合は式 1.12 より $I_{\mathrm{u,v}} = H_{\mathrm{u}} = H_{\mathrm{v}}$ となります。

式 1.13 は以下のようにも書き換えられます。

$$I_{\mathrm{u,v}} = \sum_{i=\mathrm{a,c,g,t}} \sum_{j=\mathrm{a,c,g,t}} P_{\mathrm{u}:i,\mathrm{v}:j} \log_2 \frac{P_{\mathrm{u}:i,\mathrm{v}:j}}{P_{\mathrm{u}:i} P_{\mathrm{v}:j}} \tag{1.14}$$

では図 1.12 の位置 u,v の塩基について実際に相互情報量を計算してみましょ

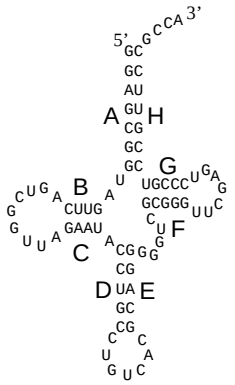


図 1.13　大腸菌 tRNA の二次構造の例

う。(a)〜(c) のどれについても位置 u,v に 4 種類の塩基 a,c,g,t が均等に出現するため、$H_{\mathrm{u}} = H_{\mathrm{v}} = 2$ です。次に結合エントロピーと相互情報量を計算してみましょう。(a) のケースでは、16 種類の a-a,a-c,a-g,...,t-c,t-g,t-t の全ての組み合わせの塩基の組が均等に出現するため、$H_{\mathrm{u,v}} = 4$ です。従って、$I_{\mathrm{u,v}} = 0$ となります。次に (b) のケースでは、位置 u,v について 8 種類の組み合わせの塩基の組が出現しているため、$H_{\mathrm{u,v}} = 3$ であり、$I_{\mathrm{u,v}} = 1$ となります。(c) のケースでは、位置 u,v について 4 種類の組み合わせの塩基の組しか出現していません。従って、$H_{\mathrm{u,v}} = 2$、$I_{\mathrm{u,v}} = 2$ となります。

(a) のケースでは、u の位置の塩基の出現と、v の位置の塩基の出現に関連は全くなく、相互情報量が 0 になっています。一方 (c) のケースでは、u の位置の塩基が決まれば、v の位置の塩基は完全に決まっています。つまり、u と v の位置の塩基に強い相関が見られ、相互情報量は最大値である 2 になっています。(b) の場合は、u の位置の塩基が決まると、v の位置の塩基は 2 種類のうちの 1 つに限定され、相互情報量は 1 になり、(a) と (c) のケースの中間になります。

実際に大腸菌の tRNA の配列を相互情報量によって解析した例を示します。tRNA はコード領域のコドンとアミノ酸を対応させる RNA 分子です。その二次構造は図 1.13 のようになっていることがわかっています。大腸菌で長さが 76〜77 塩基のものをアラインメントにかけたものを表 1.3 に示します。

これらの tRNA 配列の全位置の組み合わせに対して、相互情報量を計算したのが、表 1.4 です。図 1.13 で見られた塩基が対合している領域 (A-H, B-C, D-E, F-G) において相互情報量が高くなっているのが分かります。

1.7　塩基間の相互作用の統計的有意性

最後に塩基間の従属関係の強さではなく、統計的な有意性を調べる方法につ

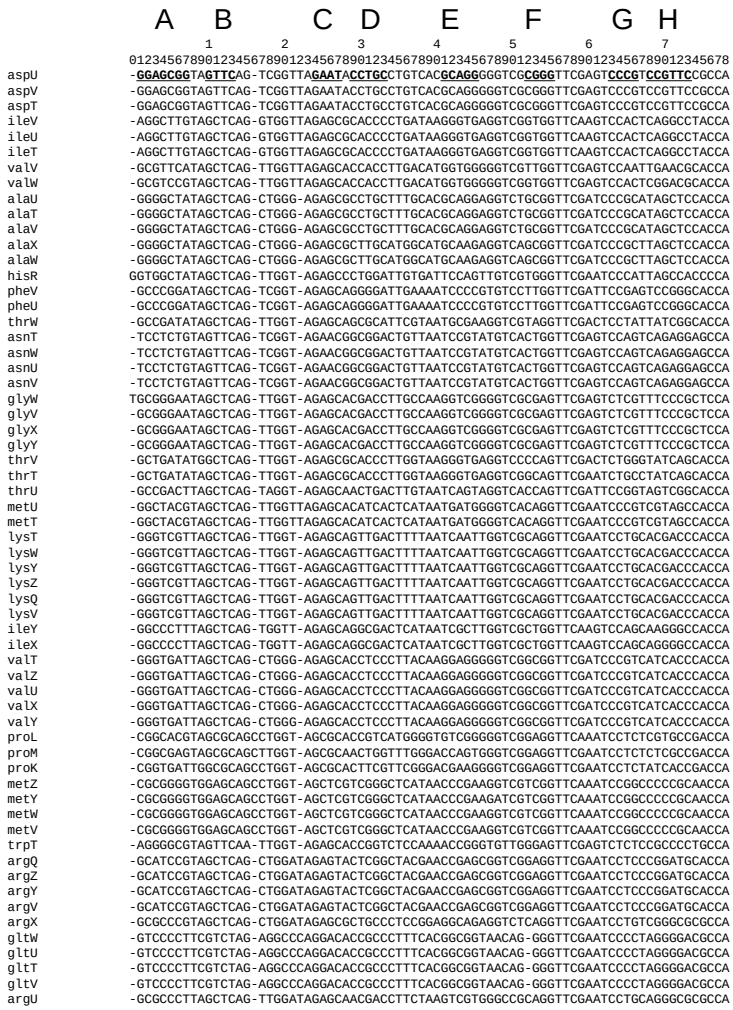


表 1.3　大腸菌 tRNA の配列

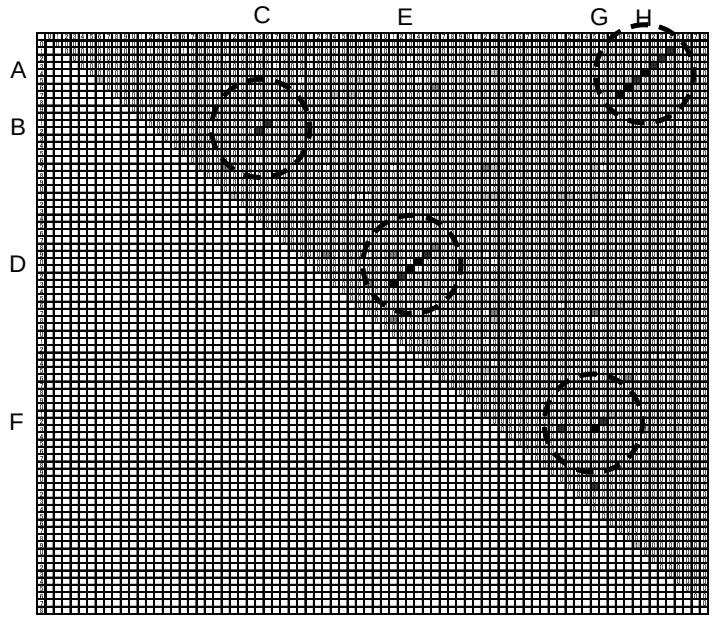


表 1.4　大腸菌 tRNA の相互情報量

表 1.3 に掲載した tRNA 配列の全ての位置間の相互情報量を計算した結果を示した。一番左の列と、一番上の行に組となる位置を示し、対応する相互情報量 (1 の位のみ) をマスの中に書いた。相互情報量の大きさに応じてマスの色を濃くした。表の欄外には図 1.13 に対応する位置をラベルし、対合している箇所 (ステム構造) を丸の点線で囲った。

いて触れておきます。これは節 1.4 で説明した方法と同一の理論的背景を持っており、詳しくは B.4 にまとめておきました。

今仮説 H_0 を、位置 u と v の塩基はそれぞれ独立に出現するという命題とします。

$$H_0 : P_{\mathrm{u,v}}(i,j) = P_{\mathrm{u}}(i)P_{\mathrm{v}}(j)$$

但し、$P_{\mathrm{u,v}}(i,j)$ はある一本の配列において、位置 u に塩基 i が、位置 v に塩基 j が観測される確率とし、実際に観測された割合 $P_{\mathrm{u}:i,\mathrm{v}:j}$ とは区別して考えます。同様に、$P_{\mathrm{u}}(i)$、$P_{\mathrm{v}}(j)$ はそれぞれ位置 u に置いて塩基 i が観測される確率と位置 v において塩基 j が観測される確率です。

次に対立仮説 H_1 を、位置 u と v に出現する塩基の間には何らかの相関があるという命題とします。

$$H_1 : P_{\mathrm{u,v}}(i,j) \neq P_{\mathrm{u}}(i)P_{\mathrm{v}}(j)$$

仮説 H_0 が正しいとすれば、以下の式によって算出される χ^2 値は自由度 9 の χ^2 分布に従います [8]。

$$\chi^2\text{値} = \sum_{i=\mathrm{a,c,g,t}} \sum_{j=\mathrm{a,c,g,t}} \frac{(N_{\mathrm{u}:i,\mathrm{v}:j} - \frac{N_{\mathrm{u}:i}N_{\mathrm{v}:j}}{N})^2}{\frac{N_{\mathrm{u}:i}N_{\mathrm{v}:j}}{N}} \tag{1.15}$$

但し、N は解析対象の配列の数、$N_{\mathrm{u}:i}$ は位置 u に塩基 i が存在する配列の本数、$N_{\mathrm{v}:j}$ は位置 v に塩基 j が存在する配列の本数、$N_{\mathrm{u}:i,\mathrm{v}:j}$ は位置 u に塩基 i が存在し、位置 v には塩基 j が存在する配列の本数です。以下の式が成り立つことに注意しましょう。

$$P_{\mathrm{u}:i} = \frac{N_{\mathrm{u}:i}}{N}, \quad P_{\mathrm{v}:j} = \frac{N_{\mathrm{v}:j}}{N}, \quad P_{\mathrm{u}:i,\mathrm{v}:j} = \frac{N_{\mathrm{u}:i,\mathrm{v}:j}}{N}$$

N が十分大きいとき、式 1.15 は近似的に以下の対数尤度比統計量に近づきます (B.4 参照)。

$$\text{対数尤度比統計量} = 2N \sum_{i=1}^{r} \sum_{j=1}^{c} P_{\mathrm{u}:i,\mathrm{v}:j} \log \frac{P_{\mathrm{u}:i,\mathrm{v}:j}}{P_{\mathrm{u}:i}P_{\mathrm{v}:j}} \tag{1.16}$$

式 1.16 は配列の本数と相互情報量の積の 2 倍になっています。これは相互作用の程度が同じでも、配列の本数が多ければそれだけ有意になることを意味しています。

演習問題

1.1 複数の DNA 配列上のある特定の位置において、A が 50%、C が 50%、G、T が 0%観測されているときのこの位置のエントロピーを求めましょう。

1.2 対照となる塩基の頻度を $B_{\mathrm{a}} = 0.3, B_{\mathrm{c}} = 0.2, B_{\mathrm{g}} = 0.2, B_{\mathrm{t}} = 0.3$ として、与えられた位置における塩基 i の個数 N_i が

(a) $N_{\mathrm{a}} = 50, N_{\mathrm{c}} = 30, N_{\mathrm{g}} = 10, N_{\mathrm{t}} = 10$

(b) $N_{\mathrm{a}} = 500, N_{\mathrm{c}} = 300, N_{\mathrm{g}} = 100, N_{\mathrm{t}} = 100$

のときの増加情報量、χ^2 値をそれぞれ求めましょう。

1.3 下の位置 u,v の塩基間の相互情報量を求めましょう。

```
u v
---
A A
A A
A G
A T
C C
C C
C G
C T
```

1.4 相互情報量を表す式 1.14 から式 1.13 を導出しましょう。

第 2 章
アラインメントアルゴリズム

アラインメントは複数の塩基配列またはアミノ酸配列から類似性の高い領域を探して揃える手法です。塩基やアミノ酸配列を解析する上でアラインメントは基本となる手法であり、その応用範囲は、遺伝子およびタンパク質の機能予測、モチーフの抽出、系統樹作成とかなり広く、バイオインフォマティクスの分野では非常に重要な位置を占めています。そこで本章では、アラインメントの基本的理論および実装方法について論じます。

2.1 アラインメントとは？

アラインメントとは核酸配列やアミノ酸配列の似ている部分を揃えることです。図 2.1 をご覧下さい。左側に 2 本の配列があり、配列を上下で比較すると、同一の塩基となっている部分は先頭の 1 塩基だけです。ところが上の配列の最初と 2 番目の塩基の間にギャップ (gap, ”-”で表します) を入れると、同一となっている塩基は 3 つに増えます。このようにアラインメントは、ギャップを入れることによって複数の配列の類似性を高める手法だと言えます。アラインメントを使うと、複数の配列において保存されている部分を特定したり、配列同士

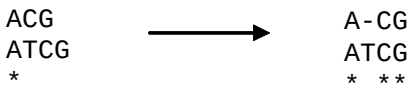


図 2.1　簡単なアラインメントの例
左側にアラインメント前の塩基配列、右側にアラインメント後の塩基配列を載せた。”-”はギャップ。上下で塩基が同一になった箇所を”*”で示した。

図 2.2　アラインメントの塩基の一致・不一致とギャップ
上下で塩基が一致している箇所を”*”、一致していない箇所を”-”で表した。

の進化関係を推定したりすることができます。

ギャップは一般に塩基の挿入または欠損を表します。例えば図 2.2 では、配列 2 の 6 の位置の塩基が欠損して配列 1 になったと考えるか、あるいは配列 1 の 5 番目の次の位置に A が挿入されて配列 2 になったと考えます。また塩基の不一致は塩基置換を表します。図 2.2 では、配列 1 の 3 番目の塩基 T が塩基置換を起こして配列 2 の 3 番目の塩基 C になったと考えるか、あるいは、配列 2 の 3 番目の塩基 C が塩基置換を起こして配列 1 の 3 番目の塩基 T になったと考えます。

2 本の配列のアラインメントをペアワイズアラインメント (pairwise alignment)、3 本以上の配列のアラインメントをマルチプルアラインメント (multiple alignment) と呼びます。本章ではペアワイズアラインメントを扱います。

また配列全体をそろえるようなアラインメントをグローバルアラインメント (global alignment、大域的アラインメント)、一部分だけをそろえるようなアラインメントをローカルアラインメント (local alignment, 局所的アラインメント) といいます。

では 2 本の配列が与えられたとき、類似性が最大になるようにアラインメントするにはどうすればいいでしょうか？これがまさに本章の目的であり、２つの部分に分けて考えることができます。

1. アラインメントによる類似性の高さをどのように評価するか？
2. その評価基準で類似性を最大にするにはどうすればいいか？

まずは 1 について考えてゆきましょう。

2.2　アラインメントの評価方法

良いアラインメントとは何でしょうか？アラインメントの類似性の高さをどのように評価すればいいでしょうか？単純な評価体系はアラインメントを行って２つの塩基配列を比較したとき、位置ごとに

1. ２つの塩基が一致しているときの点（正の値）
2. ２つの塩基が一致していないときのペナルティー（負の値）
3. ギャップを入れたときのペナルティー（負の値）

を定義することです。そして、各アラインメント位置の評価が独立である (つまり、ある位置の評価が、他の位置の評価に影響されない) と考え、これらの点を全ての位置で合計したものをスコアとして考えます。

例えば塩基が一致したときの点を+10 点、一致していないときのペナルティーを-7 点、ギャップのペナルティーを-5 点と定義します。図 2.2 に示すようなアラインメントを得た場合、アラインメントの点は 1 の位置の点＋ 2 の位置の点＋ ･･･+9 の位置の点になります。２つの塩基が一致しているところは 7 箇所、一致していないところは 1 箇所、ギャップが 1 箇所なので、このアラインメントのスコアは、$10 \times 7 + (-7) \times 1 + (-5) \times 1 = 58$ となります。

さてアラインメントのスコア体系がこのように定義できたので、いよいよ最適なアラインメントを求める方法について少しずつ触れていきたいと思います。

2.3　アラインメントのグラフ表現

アラインメントをグラフで表現すると、最適なアラインメントを求める方法を比較的容易に理解することができるでしょう。実は全てのアラインメントの状態は図 2.3 のように左上から右下に向かう経路で表すことができるのです。

アラインメントしたい 2 本の配列を図 2.3 の左のように縦と横に並べます。そしてノード (○) を塩基配列の先頭と最後、そして塩基配列の間に長方形の平面を敷き詰めるように並べます。長さ 3 と 4 の塩基配列の場合、$(3+1) \times (4+1) = 20$ 個のノードを敷き詰めます。するとアラインメントは、一番左上のノードから一番右下のノードまでの経路で表すことができます。このとき、１つ右斜め下、１つ右、１つ下の 3 方向の移動しか許されません。

すなわち 2 本の配列は、

1. 右斜め下のノードに移動するときは、移動のときに縦方向および横方向の位置で通過する配列 1 と 2 の両方の塩基がアラインメントされる。
2. 右方向のノードに移動するときは、ギャップと配列 2 の横方向で通過する塩基がアラインメントされる

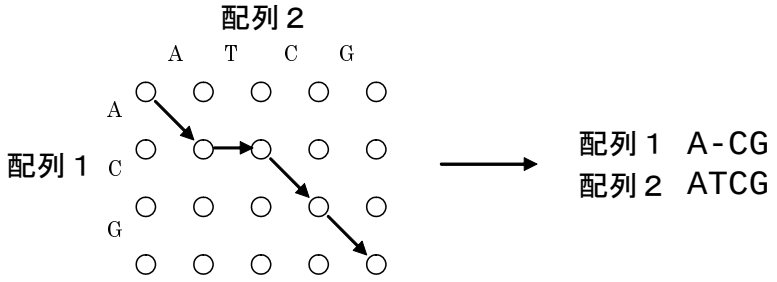


図 2.3　アラインメントのグラフ表現

左側にノードを通過する経路、右側にその経路に対応するアラインメントを示した。一番左上のノードから１つ右下のノードに移動するときは、配列１の最初の塩基 A を縦方向に通過し、配列 2 の最初の塩基 A を横方向に通過するので、A 同士がアラインメントされる。その次に右方向に１つ移動するときは、配列 2 の次の塩基 T のみを横方向に通過するため、ギャップと T がアラインメントされる。

3. 下方向のノードに移動するときは、配列 1 の縦方向で通過する塩基とギャップがアラインメントされる

という規則でアラインメントされます。

すると最適なアラインメントを見つけるという問題は、最適な経路を見つけるという問題に帰着することができるのです。

2.4 最適アラインメントの数式化

ここで最適アラインメントとは何かを再確認しておきましょう。最適アラインメントは最大のスコアを持つアラインメントと考えます。グラフを使って言いかえると、「右下のノードに至る最適の経路」となります。では、右下のノードに至る最適の経路をどのように求めればいいでしょうか？

今長さ m の配列と、長さ n の配列をアラインメントすることを考えます。ノードは全部で $(m+1)\times(n+1)$ 個並ぶことになります。一番上の行を 0 行目、一番左の列を 0 列目と数え、i 行目、j 列目の位置にあるノードを i,j と表します。一番右下のノードは m,n と表されることになります。ノード i,j に至る直前のノードは必ず左 $(i,j-1)$、上 $(i-1,j)$、斜め上 $(i-1,j-1)$ の 3 つのうちのどれかになることに注意してください。また各アラインメント位置の評価が独立であるというスコア体系を使用するため (塩基の一致や不一致、ギャップに対して決まったスコア・ペナルティーが付けられる)、$(i,j-1)$、$(i-1,j)$ もしくは $(i-1,j-1)$ から (i,j) にいくときの追加スコアやギャップペナルティはそれより前の経路に関係なく一定です。すると、右下のノードに至る最大スコアは

- $(i,j-1)$ までの最大スコアにギャップペナルティを足したもの
- $(i-1,j)$ までの最大スコアにギャップペナルティを足したもの
- $(i-1,j-1)$ までの最大スコアにアラインメントスコア (塩基が一致したときの点・不一致のときのペナルティー) を足したもの

の 3 つのうち、最大のものになります (図 2.4)。実はこれは $i>0, j>0$ となる任意のノードに当てはまることです。ただし、一番左端および上端の最大スコアは 0 点と定義しておきます。このようにすると、アラインメントの左端で最初に現れるギャップ領域のペナルティーは 0 になります[*1)]。

数式で表現すると、

*1) このままではアラインメントの右端のギャップは 0 にはならない。演習問題 2.7 参照。

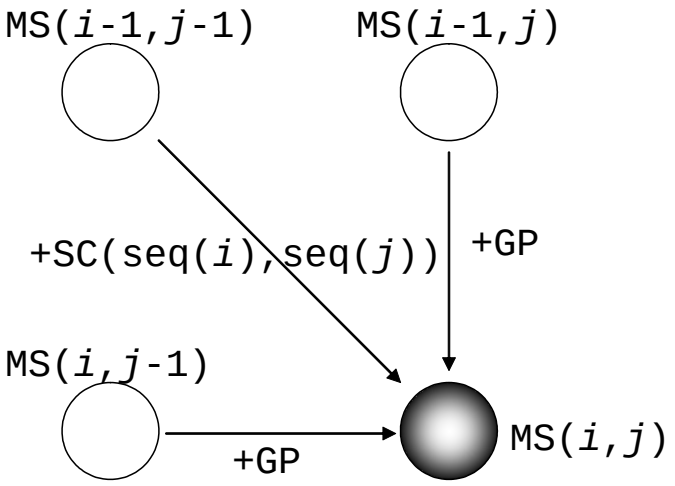


図 2.4　最大スコアの求め方

ノード (i, j) に至るまでの最大スコア MaxScore(i, j) の求め方を示した。ノード (i, j) に至るまでの経路は左、上、左斜め上と 3 つあるため、これら 3 方向について最大スコアはいくつになるか比較すればよい。MaxScore は MS,GapPenalty は GP, Score は SC とそれぞれ略記。

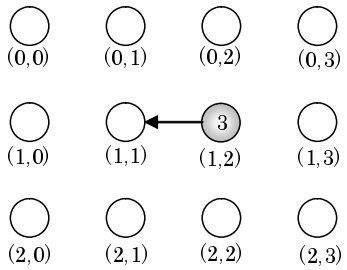


図 2.5　各ノードが持つべき情報
ノードの座標を各ノードの下に示した。本文中で注目しているノード (1,2) に最大スコアと直前のノードの方向の情報を加えた。

$$\mathrm{MaxScore}(i,j) = \begin{cases} 0 \text{ if } i=0 \text{ or } j=0 \\ \max \begin{cases} \mathrm{MaxScore}(i-1,j) + \mathrm{GapPenalty} \\ \mathrm{MaxScore}(i,j-1) + \mathrm{GapPenalty} \\ \mathrm{MaxScore}(i-1,j-1) \\ \qquad +\mathrm{score}(\mathrm{seq1}(i-1), \mathrm{seq2}(j-1)) \end{cases} \end{cases} \tag{2.1}$$

となります。但し、$\mathrm{MaxScore}(i,j)$ は (i,j) に至るときの最大スコア、GapPenalty はギャップに対して与えられるペナルティー (<0)、$\mathrm{Score}(b_1,b_2)$ は塩基 b_1, b_2 がマッチ・ミスマッチしたときのスコア・ペナルティー、$\mathrm{seq}(i)$ は配列 seq の i 番目の塩基を表します。但しここでは最初の塩基を $\mathrm{seq}(0)$ から数えます。2 本の配列をアラインメントしたときの最大スコアは $\mathrm{MaxScore}(m,n)$ と表されます。

2.5　アラインメントアルゴリズムの実装

アラインメントを行う上で中核となるのは式 2.1 です。これを実際にプログラムとして実装する方法を説明してゆきます。

2.5.1　ノードを扱う構造体

まず、各ノードの持つべき情報は、

1. そのノードに到達したときの最大スコア
2. 最大スコアになるときの直前のノードの方角

の 2 つです。例えば図 2.5 のノード (1,2) に注目してみましょう。このノードが持つべき情報は、ここに到達するときの最大スコア（例えば 3 点）と、ここに到達するときの直前のノードの方角（例えば左）です。

各ノードの情報は以下のように構造体で表すといいでしょう。

```
struct {
   int score; /* そのノードに至ったときの最大スコア */
   int direction; /* そのノードに到達して最大スコアになったときの
                     直前のノードの方角 */
} score_path_matrix[MAX_SEQ_LEN][MAX_SEQ_LEN];
  /* MAX_SEQ_LEN は扱える最大の配列長さ */
```

2.5.2 最大アラインメントスコアの関数

では実際にノード (i, j) に到達するときの最大スコアを計算する関数 `find_max_score(int i, int j)` を作成してみましょう。

今変数 `max_score` に最大スコア、`max_dir` に最大スコアになるときの直前のノードの方向を入れることにします (上を V、左を H、斜め左上を D とします。V,H,D は実際は`#define` で定義された数値 0,1,2 です。)。

まず上端は 0 なので、

```
   if(i == 0){ max_score = 0; max_dir = H;}
```

となります。そして左端も同様に

```
   else if(j == 0){ max_score = 0; max_dir = V; }
```

でしょう。

さて、それ以外の場合は再帰を使うことになります。式 2.1 の max の項を実装して、以下のようにします。

```
   else {
     score_tmp[V] = find_max_score(i - 1, j) + Gap_Penalty;
     /* (i-1,j) までの最大スコアにギャップペナルティを足したもの */
     score_tmp[H] = find_max_score(i, j - 1) + Gap_Penalty;
     /* (i,j-1) までの最大スコアにギャップペナルティを足したもの */
     score_tmp[D] = find_max_score(i - 1, j - 1) +
       score(seq1[i - 1], seq2[j - 1]);
     /* (i-1,j-1) までの最大スコアにアラインメントスコアを
        足したもの */

     max_dir = find_max_elem(score_tmp, NDIR);
     /* score_tmp[V], score_tmp[H], score_tmp[D] のうちで最大の
```

```
        方向 (V,H,D) を求める */
    max_score = score_tmp[max_dir];
    /* 上記で求めた方向に対応する最大スコアを求める */
  }
  score_path_matrix[i][j].score     = max_score;
    /* (i,j) における最大スコアを記録 */
  score_path_matrix[i][j].direction = max_dir;
    /* (i,j) が最大スコアになるときの直前のノードの方向 */
  return max_score;
    /* 最大スコアを返す */
}
```

但し、上のプログラムで score は塩基がマッチしたときのスコアを求める関数 (演習問題 2.1)、find_max_elem はスコアが最大になる方向を求める関数です (演習問題 2.2)。そしてプログラムの先頭の方で次のいくつかの定数の定義を行います。

```
#define H 0   /* 横方向 */
#define V 1   /* 縦方向 */
#define D 2   /* 斜め方向 */
#define NDIR 3 /* 方向数 */
#define Gap_Penalty -5 /* ギャップペナルティ */
#define MAX_SEQ_LEN 100 /* プログラムで扱える最大の配列の長さ */
```

2.5.3 プログラムの実行

プログラムを実際に実行するには、配列 seq1 と seq2 に実際の核酸配列を入れて、find_max_score を呼び出すだけです。これで最大スコアを求めることができます (演習問題 2.3)。

```
#include <string.h>

main(){
  int i,j;
  strcpy(seq1, "acg");
  strcpy(seq2, "aacg");
  printf("%d\n", find_max_score(3, 4));
}
```

score

配列2

配列1	0	1	2	3	4
0	0	0	0	0	0
1	0	10	5	0	−5
2	0	5	3	15	10
3	0	0	−2	10	25

direction

配列2

配列1	0	1	2	3	4
0	H	H	H	H	H
1	V	D	H	H	H
2	V	V	D	D	H
3	V	V	V	V	D

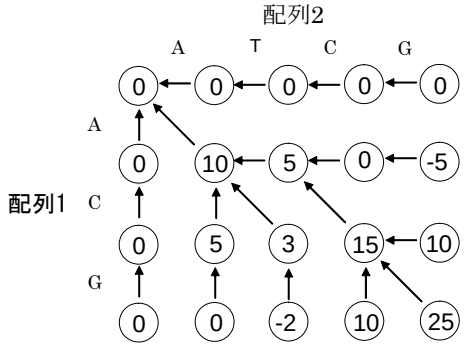


図 2.6　各ノードに含まれる最大スコアおよび直前のノードの情報

本文 2.5 のプログラムを実装したときに、最終的にできる score_path_matrix[][].score (上段) および score_path_matrix[][].direction (中段) の情報。下段に上段と中段の情報を統合したものを示した。なお、中段の (0,0) の位置が H になっているが、これはプログラムの実装上の問題で、本質的な意味はない。

2.5.4　各ノードの情報

プログラムを実行させると、score_path_matrix[][].score および score_path_matrix[][].direction の中味は最終的に図 2.6 の上段および中段に示すような内容になるはずです。また上段と中段の情報をもとに、下段には各ノードが持つ最大スコアおよびその最大スコアを与える直前のノードの方向を各ノードに対して示しておきました。２つの配列”acg”と”atcg”のアラインメントのスコアは一番右下のノードの情報より 25 であることが分かります。また一番右下のノードから直前の方向を辿っていくと、アラインメントを求めることができます。

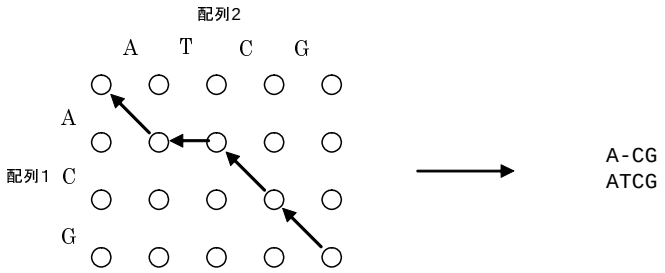


図 2.7　通過した経路からアラインメントへ

2.5.5 アラインメントの表示

それではいよいよアラインメント情報 score_path_matrix から実際のアラインメントを得る部分のアルゴリズムを考えましょう (演習問題 2.4)。簡単に言うと、一番右下のノードから score_path_matrix の direction をたどって左上に到達するまでに通過した経路でアラインメントを決めます (図 2.7)。詳細なアルゴリズムは以下の通りです。

1. 空の配列 a_seq1[], a_seq2[] を用意します。
2. まず自分が見ているノード (i, j) を一番右下のノードに設定します。
3. score_path_matrix[i][j].direction の値をみて (図 2.8(a))、
   (a) H なら、a_seq1 に GAP(-) を、a_seq2 に seq[j-1] を追加します (図 2.8(b))。その後、j の値を 1 つ引きます。
   (b) V なら a_seq1 に seq[i-1] を、a_seq2 に GAP(-) を追加します (図 2.8(c))。その後、i の値を 1 つ引きます。
   (c) D なら a_seq1 に seq[i-1] を、a_seq2 に seq[j-1] を追加します (図 2.8(d))。その後、i,j の値を 1 つ引きます。
4. $i > 0$ または $j > 0$ なら 3 に戻り、現在のノード (i, j) に対して同じような操作をします。
5. 現在のノードが (0,0) に到達した後、a_seq1 と a_seq2 の中の文字列を反転させれば出来上がりです。

2.5.6 計算の重複の回避

扱う配列が長くなると、アラインメントにかかる時間は急激に増大します。そこで節 A.3 を参考にアラインメントを高速化する手法を考えましょう。関数 find_max_score では以下のように二次元配列に最大スコアとその経路を記録していました。

```
score_path_matrix[i][j].score     = max_score;
  /* (i,j) における最大スコアを記録 */
score_path_matrix[i][j].direction = max_dir;
  /* (i,j) が最大スコアになるときの直前のノードの方向 */
```

一方で find_max_score は同じ (i, j) に対しても何回も呼ばれます。そこで、score_path_matrix[i][j] に値が記録されていれば、2 回目の呼び出しからはそれを使うことにより計算時間を大幅に節約することができます (演習問題 2.5)。

```
if(score_path_matrix[i][j].score != INVALID){
   return score_path_matrix[i][j].score;
```

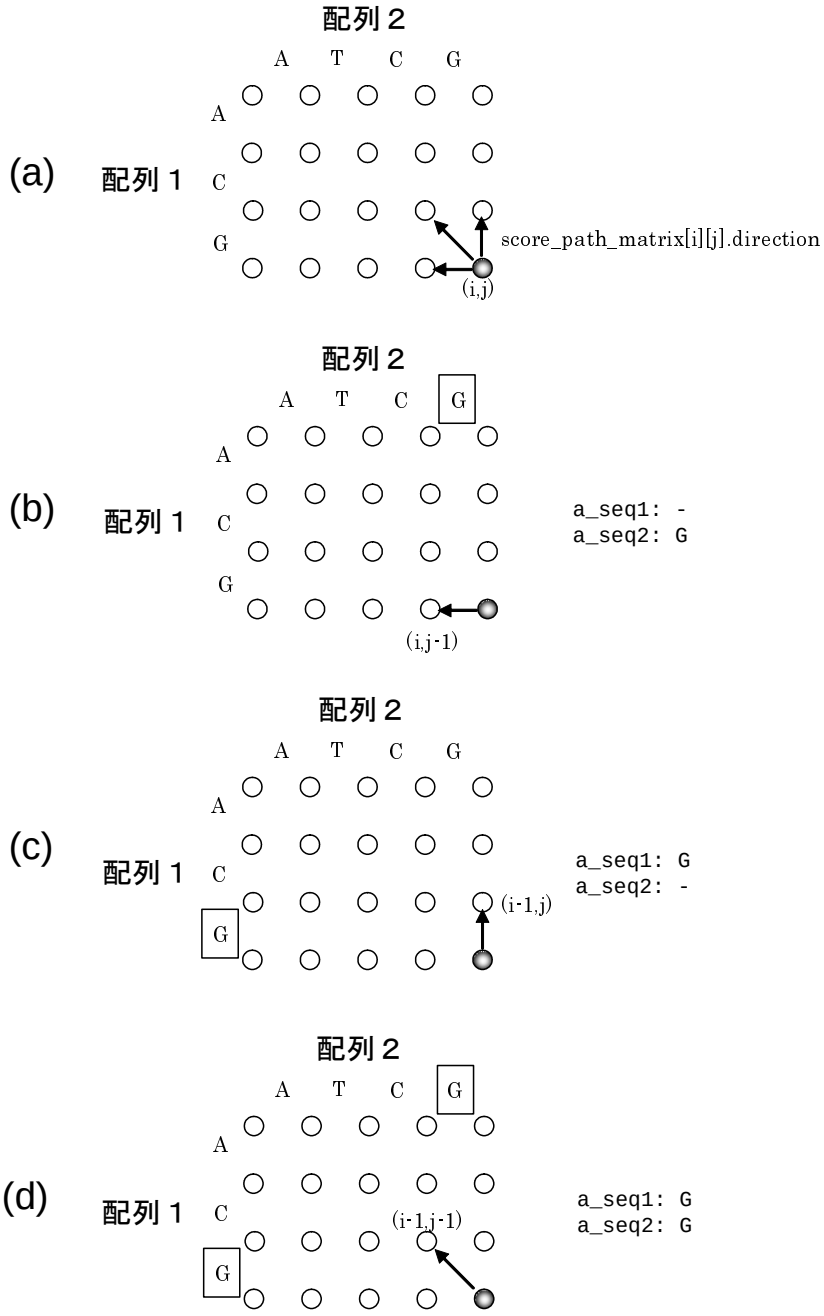


図 2.8　通過した経路からアラインメントを求めるアルゴリズム

```
}
```

このテクニックを使うためには、main のところで最初に score_path_matrix の値を初期化する必要があります。

```
#define INVALID 1000

/* 初期化 */
  for(i = 0;i < 30;i ++)
    for(j = 0;j < 30;j ++)score_path_matrix[i][j].score = INVALID;
```

これまで見てきたように式 2.1 をそのまま実装すると、関数は再帰関数となります。しかし、再帰関数を使わずにアラインメントを実装することも可能です。

MaxScore(i, j) を求めるためには、MaxScore$(i-1, j)$、MaxScore$(i, j-1)$、MaxScore$(i-1, j-1)$ の 3 つの値が必要です。再帰関数では、MaxScore(i, j) が呼び出された後、、MaxScore$(i-1, j)$、MaxScore$(i, j-1)$、MaxScore$(i-1, j-1)$ が呼び出され、その後で MaxScore(i, j) を計算していました。この呼び出しのステップを省き、i, j の値が小さいほうから順に MaxScore$(0, 0)$, MaxScore$(0, 1)$, MaxScore$(0, 2)$, $\cdots$ MaxScore(i, j) を求めていけば、再帰計算を使わずに、かつ計算の重複を起こすことなくアラインメントを行うことができます。以下にその実装例を示します。

```
int make_score_path_matrix(int seq1_len, int seq2_len){
  int score_tmp[NDIR];
  int max_dir;
  int max_score;
  int i,j;
  for(i = 0;i <= seq1_len;i ++){
    for(j = 0;j <= seq2_len;j ++){
      if(i == 0){ max_score = 0; max_dir = H; }
      else if(j == 0){ max_score = 0; max_dir = V; }
      else {
        score_tmp[V] = find_max_score(i - 1, j) + Gap_Penalty;
        score_tmp[H] = find_max_score(i, j - 1) + Gap_Penalty;
        score_tmp[D] = find_max_score(i - 1, j - 1) +
          score(seq1[i - 1], seq2[j - 1]);
        max_dir = find_max_elem(score_tmp, NDIR);
        max_score = score_tmp[max_dir];
```

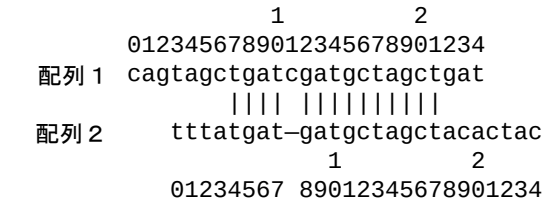


図 2.9　ローカルアラインメントの概要

配列の上下に、0 から始まる塩基番号を示した。また 2 本の配列がクリアに一致している箇所を”|”で示した。

```
    }
    score_path_matrix[i][j].score     = max_score;
    score_path_matrix[i][j].direction = max_dir;
   }
  }
  return score_path_matrix[seq1_len][seq2_len].score;
}
```

2.6　ローカルアラインメント

グロバールアラインメントは配列全体の類似性を高くするアラインメントです。一方、ローカルアラインメントでは全体を揃えるのではなく、似ている部分のみを揃えるようにします [10]。基本的な概念を図 2.9 に示します。この例では中央付近で 2 本の配列の相同性が非常に高くなっています。しかし、両端部分には相同性がほとんどありません。スコア体系にもよりますが、配列 1 の 0〜6 番目の塩基配列と、配列 2 の 0〜3 番目の塩基配列を無理にアラインメントしても、アラインメントが適切に行われたことを示すスコア (一般的には 0 より大きいスコア) が得られることが期待できません。また配列 1 の 22〜24 番目と配列 2 の 18 〜24 番目に関しても同じことが言えます。そこで、ローカルアラインメントでは、スコアが高くなりそうもない部分ではアラインメントを打ち切ってしまいます。この例でいうと、(7,4) の位置のノードより先頭に戻ってもスコアが 0 より大きくならない場合、そこでアラインメントを打ち切ります。また、ノードが持つアラインメントのスコアが最も高くなる (22,18) からアラインメントを開始します。

一般的な議論をすると基本的には、ある地点 (i, j) までアラインメントした

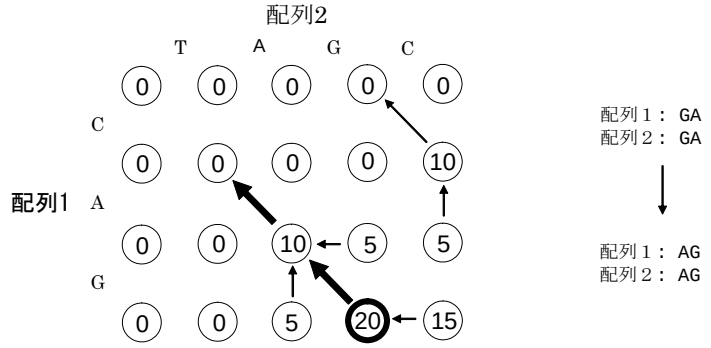


図 2.10　ローカルアラインメントの例

左に 2 本の配列”cag”と”tagc”をローカルアラインメントにかけた時の各ノードの最大スコアおよびそれを与える直前のノードの方向を示した。アラインメント開始ノード (最も大きなスコアを持つノード) を太い○で表し、またアラインメントされた経路を太い矢印で表した。右に実際に生成されるアラインメントを示した。

最大スコア $\mathrm{MaxScore}(i,j)$ が 0 以下の値になっているとき、そこまでのアランメントは意味がないので、アラインメントを打ち切ります。アラインメントをしない状態のスコアを 0 として、それを選択肢の 1 つに組み込むのです。これを数式で表現すると、

$$\mathrm{MaxScore}(i,j) = \begin{cases} 0 \text{ if } i=0 \text{ or } j=0 \\ \max \begin{cases} 0 \\ \mathrm{MaxScore}(i-1,j) + \mathrm{GapPenalty} \\ \mathrm{MaxScore}(i,j-1) + \mathrm{GapPenalty} \\ \mathrm{MaxScore}(i-1,j-1) \\ \qquad +\mathrm{score}(\mathrm{seq1}(i-1), \mathrm{seq2}(j-1)) \end{cases} \end{cases} \tag{2.2}$$

となります。

図 2.10 に各ノードが持つ最大スコアと直前のノードの方向、およびそこから得られるアラインメントの例を示しておきます。最大スコアが 0 になっているノードでは直前のノードが示されておらず、アラインメントが打ち切られているのが分かると思います。またアラインメントが $\mathrm{MaxScore}(i,j)$ が最も高い位置から始まっていることが分かります。

2.7　より高度なギャップペナルティーの計算

DNA 分子が進化するとき、挿入や欠損はまとまって起こることがよくありま

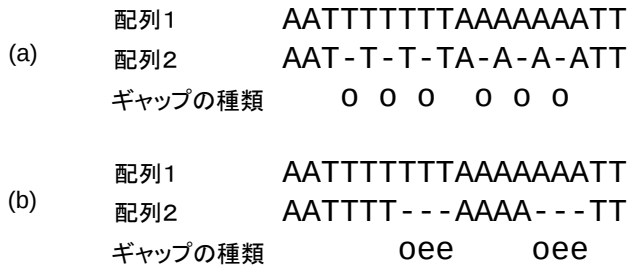


図 2.11　ギャップ開始ペナルティーとギャップ伸長ペナルティー

同一の配列の組に対して、2 種類のアラインメント (a)(b) を示した。ギャップ開始ペナルティーの対象となる位置に”o”のラベルを、ギャップ伸長ペナルティーの対象となる位置に”e”のラベルを付けた。

す。これを考慮に入れると、図 2.11(a) のようなアラインメントより、(b) のようなアラインメントに対して高いスコアをつけるべきでしょう。これを実現するのが、ギャップ開始ペナルティー ($\mathrm{GapPenalty_{open}}$) とギャップ伸長ペナルティー ($\mathrm{GapPenalty_{ext}}$) を使ったアフィンギャップペナルティーと呼ばれるスコア体系です [11]。このスコア体系では、ギャップが連続しているとき、最初のギャップのペナルティーがギャップ開始ペナルティーとなり、それ以降がギャップ伸長ペナルティーとなります。$\mathrm{GapPenalty_{open}} < \mathrm{GapPenalty_{ext}} < 0$ と設定すれば、上記の例では下のほうがスコアが高くなります。例えば $\mathrm{GapPenalty_{open}} = -5$, $\mathrm{GapPenalty_{ext}} = -1$ にすれば、図 2.11(a) のアラインメントではギャップペナルティの合計は、$\mathrm{GapPenalty_{open}} \times 6$ で-30、(b) のアラインメントでは、$\mathrm{GapPenalty_{open}} \times 2 + \mathrm{GapPenalty_{ext}} \times 4$ で-14 になり、下の方がギャップペナルティーの合計が低くなり、スコアが高くなります。このスコア体系を使うと、ギャップペナルティーが一定ではなく経路に依存して変化するため、式 2.1 を使うことはできなくなります。

そこで、$\mathrm{MaxScore}(i,j)$ を再帰式を使って定式化するために、以下の 3 つの式を定義します。

MaxScore$_{\mathrm{H}}(i,j)$ 直前のノードが左となるときの (i,j) における最大スコア
MaxScore$_{\mathrm{V}}(i,j)$ 直前のノードが上となるときの (i,j) における最大スコア
MaxScore$_{\mathrm{D}}(i,j)$ 直前のノードが左斜め上となるときの (i,j) における最大スコア

これらの式を使うと

$$\mathrm{MaxScore}(i,j) = \max \begin{cases} \mathrm{MaxScore_H}(i,j) \\ \mathrm{MaxScore_V}(i,j) \\ \mathrm{MaxScore_D}(i,j) \end{cases}$$

と表すことができます。では $\mathrm{MaxScore_H}(i,j)$、$\mathrm{MaxScore_V}(i,j)$、$\mathrm{MaxScore_D}(i,j)$ はさらにどのように表すことができるでしょうか？まず、$\mathrm{MaxScore_H}(i,j)$ について考えましょう。アラインメントで挿入の直後に欠損が起こったり、欠損の直後に挿入が起こることはないと仮定すると (つまり全ての塩基 b_1,b_2 に対して、$\mathrm{score}(b_1,b_2) > \mathrm{GapPenalty_{open}} \times 2$)、ノード (i,j) の直前に通るノードが $(i,j-1)$ のときに最大スコアになる経路は、$\mathrm{MaxScore_H}(i,j)$ は図 2.12 に示すように 2 通りになります。

$(i-1,j-2) \rightarrow (i,j-1)$ を通ってくるときは、それ以前の経路に関わらず $(i,j-1) \rightarrow (i,j)$ のギャップペナルティーは必ず $\mathrm{GapPenalty_{open}}$ になります。従って $(i-1,j-2) \rightarrow (i,j-1)$ を通るときの $(i,j-1)$ までの最大スコアは $\mathrm{MaxScore_D}(i,j-1)$ なので、(i,j) までの最大のスコアは $\mathrm{MaxScore_D}(i,j-1) + \mathrm{GapPenalty_{open}}$ で表されます。一方 $(i,j-2) \rightarrow (i,j-1)$

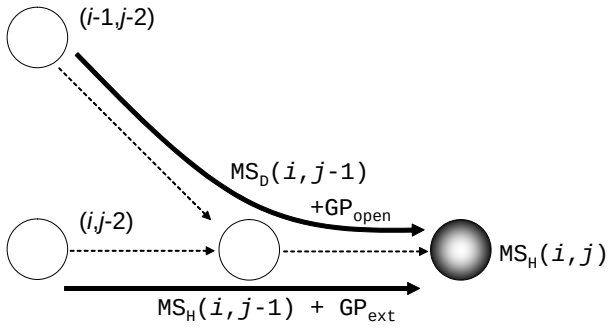


図 2.12　より高度な GapPenalty を用いたときの最大スコア $\mathrm{MaxScore}_{\mathrm{H}}(i,j)$ の計算
$(i,j-1) \rightarrow (i,j)$ の経路を通って (i,j) に到達するときの最大スコア $\mathrm{MaxScore}_{\mathrm{H}}(i,j)$ を求めるときは、矢印で示した 2 つの経路の最大スコアを比較すればよい。$\mathrm{MaxScore}_{\mathrm{H}}(i,j)$、$\mathrm{MaxScore}_{\mathrm{D}}(i,j)$、$\mathrm{GapPenalty}_{\mathrm{open}}$、$\mathrm{GapPenalty}_{\mathrm{ext}}$ はそれぞれ MS_{H}、MS_{D}、$\mathrm{GP}_{\mathrm{open}}$、$\mathrm{GP}_{\mathrm{ext}}$ と略記。

を通ってくるときは、それ以前の経路に関わらず $(i, j-1) \to (i, j)$ のギャップペナルティーは必ず $\mathrm{GapPenalty}_{\mathrm{ext}}$ になります。従って $(i, j-2) \to (i, j-1)$ を通るときの $(i, j-1)$ までの最大スコアは $\mathrm{MaxScore}_{\mathrm{H}}(i, j-1)$ なので、(i, j) までの最大のスコアは $\mathrm{MaxScore}_{\mathrm{H}}(i, j-1) + \mathrm{GapPenalty}_{\mathrm{ext}}$ で表されます。最大スコア $\mathrm{MaxScore}_{\mathrm{H}}(i, j)$ は前者と後者のうち大きいほうなので、これを数式としてまとめると、

$$\mathrm{MaxScore}_{\mathrm{H}}(i,j) = \begin{cases} 0 \text{ if } i = 0 \text{ or } j = 0 \\ \max \begin{cases} \mathrm{MaxScore}_{\mathrm{H}}(i, j-1) + \mathrm{GapPenalty}_{\mathrm{ext}} \\ \mathrm{MaxScore}_{\mathrm{D}}(i-1, j-1) + \mathrm{GapPenalty}_{\mathrm{open}} \end{cases} \end{cases}$$

になります。同様にして、

$$\mathrm{MaxScore}_{\mathrm{V}}(i,j) = \begin{cases} 0 \text{ if } i = 0 \text{ or } j = 0 \\ \max \begin{cases} \mathrm{MaxScore}_{\mathrm{V}}(i-1, j) + \mathrm{GapPenalty}_{\mathrm{ext}} \\ \mathrm{MaxScore}_{\mathrm{D}}(i-1, j-1) + \mathrm{GapPenalty}_{\mathrm{open}} \end{cases} \end{cases}$$

が成立します。$\mathrm{MaxScore}_{\mathrm{D}}(i, j)$ の再帰式は演習問題 2.9 で求めましょう。

演習問題

2.1 本文中のプログラムで使われる int score(char a, char b) は核酸（またはアミノ酸）a と b のアラインメントスコアを計算する関数です。a と b が同一のとき 10 点、a と b が異なるとき-7 点を返すようにこの関数を作りましょう。

2.2 本文中のプログラムで使われる `int find_max_elem(int score_tmp[], int n)` は n 個の要素を持つ配列 `score_tmp` の中で最大の値を持つ要素が何番目か (0〜n-1) を返す関数です。この関数を作りましょう。

2.3 与えられた 2 本の配列より最大スコアを計算するプログラムを完成させましょう。

2.4 本文中のプログラムで使われる構造体の配列 `score_path_matrix` と seq1、seq2 の情報から `a_seq1[]`、`a_seq2[]` にアラインメント結果を入れる関数

```
void board_to_alignment(char a_seq1[], char a_seq2[], int m, int n);
```

を作りましょう。なお、m,n は (i, j) の初期値、つまり一番右下のノードの位置を表します。

2.5 最大スコアの記録による高速化を実際にアラインメントに組み込んでみましょう。

2.6 2 本の配列”cct”と”ccgt”をアラインメントしたとき、各ノードが持つ最大スコアとそのスコアを与える直前のノードがどのようになるか、図 2.6 を参考に表を書きましょう。

2.7 2.5.2 の実装では配列の左側のギャップのペナルティーは 0 になりますが、右側のギャップのペナルティーは 0 になりません。例えば

```
aaaatcgatgc
---atcgatgc
```

のギャップペナルティーは 0 ですが、

```
atcgatgcaaa
atcgatgc---
```

のギャップペナルティーは 0 になりません。右側のギャップのペナルティーも 0 になるように改良しましょう。

2.8 式 2.2 を参考にローカルアラインメントを実装しましょう。

2.9 $\mathrm{MaxScore}_D(i,j)$ の再帰式を導出しましょう。

2.10 2.9 の結果を用いて、ギャップ開始ペナルティーとギャップ伸長ペナルティーを使ったスコア計算によるアラインメントを実装しましょう。

第 3 章
隠れマルコフモデル

隠れマルコフモデルは核酸配列やアミノ酸配列の出現パターンを扱うことが可能な確率モデルで、DNA 上のシグナル配列やタンパク質のモチーフ配列のモデル化に多く使われています。そして遺伝子領域予測やタンパク質の二次構造予測などの分野で大きな成果を挙げています。ここでは隠れマルコフモデルの基本概念と実装方法について学習しましょう。

3.1 マルコフモデルとは？

多くの種のゲノム配列を眺めているとあたかも A,C,G,T の塩基がランダムに出現するように見えます。しかし実際には例えば T の次に A が来ることが少なかったり、C の次に G が来ることが少なかったりと、様々な傾向を見出すことができます。また、プロモータ領域やコード領域になると塩基の偏りが著しくなります。ではゲノム配列中の塩基の出現にはどのような法則があるのでしょうか？

これを説明する１つのモデルが、マルコフモデルです。マルコフモデルとは、状態の変化を状態とその間の遷移確率で表したものです。各状態は１つの塩基を表し、塩基配列は、通過する状態によって決定されるものと考えます。

例えば図 3.1 のモデルを見てみましょう。各状態がノード（丸）で、状態遷移が矢印で表されています。各状態 1〜5 には塩基が一意に割り当てられています。そしてマルコフモデルでは、各塩基はある状態に至ったときに、その状態に対応する塩基が出力されると考えます。また塩基配列パターンは状態間をある確率で遷移することによって出力されると考えます。するとこの図で、状態が一番左から一番右に向かって遷移すると考えると、このモデルから出力される塩基配列パターンは、ATA, ACTA, ACGA の 4 つになります。では各塩基配列パターンが出力される確率はどれくらいになるのでしょうか？今、塩基配列”ACGA”が出力される確率を考えましょう。先頭の 2 文字が AC なので、

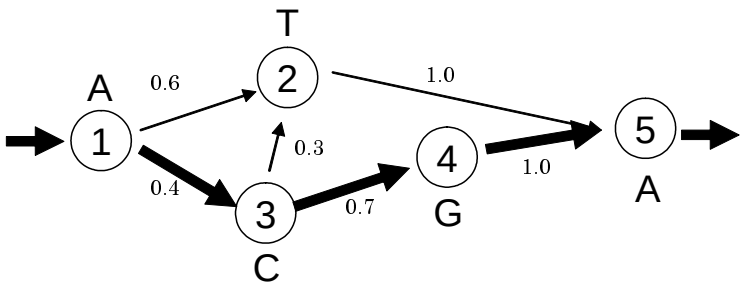


図 3.1　マルコフモデル

丸が状態、矢印が状態遷移を表す。本文中で解説している箇所を太めの矢印で表した。丸の中に状態番号を書き、その近傍にその状態が出力する塩基を書いた。また矢印の近傍に状態遷移確率を示した。

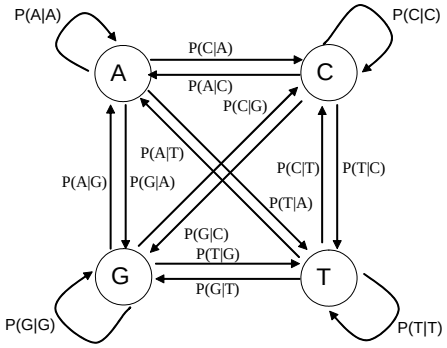


図 3.2　$P(x_i|x_{i-1})$ を表すマルコフモデル

まず A → C と遷移する確率は 0.4 です。次に C の次に G が来る確率は、C → G と遷移する確率なので、0.7 となります。同様にして最後に G → A の遷移確率は 1.0 ですから、塩基配列 ”ACGA ”の出現確率は、$0.4 \times 0.7 \times 1.0 = 0.28$ になります。

図 3.1 では、A のように 1 つの塩基を複数のノードが表しているケースがありました。しかし 1 つの塩基に対し、必ず 1 つだけノードを割り当てるようなモデルを作ることも可能です。塩基は A,C,G,T の 4 種類なので、4 つのノードが存在することになります。さらに各ノードから他の全ノードへの状態遷移を全て考えれば、図 3.2 のようなモデルができあがります。塩基の種類が決まれば、状態は一意に決まるので、ある塩基 x_{i-1} から次の塩基 x_i が出現する確率は状態を明示しなくても、$P(x_i|x_{i-1})$ と塩基だけで表すことができます。

図 3.2 は以下のような行列として表現することも可能です。

$$M = \begin{pmatrix} P(A|A) & P(C|A) & P(G|A) & P(T|A) \\ P(A|C) & P(C|C) & P(G|C) & P(T|C) \\ P(A|G) & P(C|G) & P(G|G) & P(T|G) \\ P(A|T) & P(C|T) & P(G|T) & P(T|T) \end{pmatrix}$$

多くの種のゲノム配列において、特に非コード領域の配列はマルコフモデルでよく説明することができます。

3.2　隠れマルコフモデルとは？

隠れマルコフモデルも塩基配列パターンの出現を表すことができるモデルで

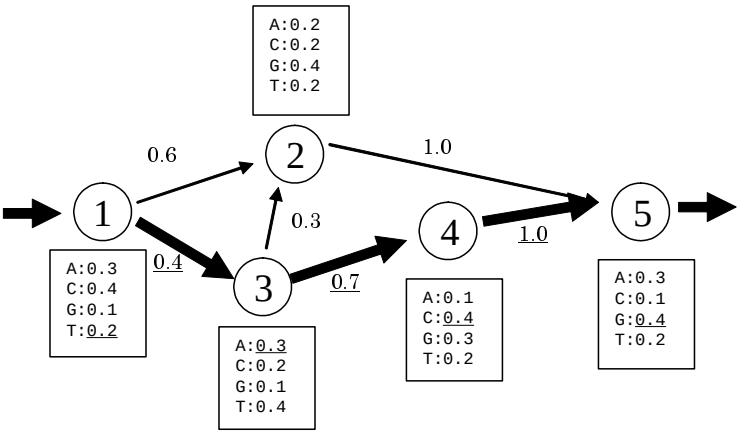


図 3.3　隠れマルコフモデル

各状態における塩基の出力確率を四角の中に書いた。本文中で触れられている経路を通過するときの確率、および経路上の各状態で出力された記号に下線を付けた。

す。マルコフモデルでは、各状態は必ず 1 つの塩基を表していました。しかし隠れマルコフモデルでは、各状態において観測される塩基は確率的に決まります。図 3.3 ように、隠れマルコフモデルでは、各状態に塩基の出現確率が付けられるのです。このモデルから出力される塩基配列は、通過した状態と、そこから確率的に出力される塩基の種類で決まります。

より形式的には隠れマルコフモデルは、状態を k, l、塩基を b として、以下の 2 つで定義されます。

$e_l(b)$	状態 l における塩基 b の観測確率
$t(k \to l)$	状態 k から状態 l への遷移確率

例えば $e_1(\mathrm{T})$ は状態 1 において塩基 T が表される確率を表し、図 3.3 ではそれは 0.2 となっています。また、$t(1 \to 3)$ は状態 1 から 3 への遷移確率を表し、図 3.3 ではそれは 0.4 になっています。

定義された隠れマルコフモデルからある経路を通って塩基パターンが出力される確率は、通過した各状態における各塩基の出力確率と、経路上の遷移確率との積になります。より形式的には、状態 $\pi = (\pi_1, \pi_2, \cdots, \pi_L)$ を通過し、長さ L の塩基配列 $x = (x_1, x_2, \cdots, x_L)$ が出力される確率は、$\left\{\prod_{i=1}^{L-1} e_{\pi_i}(x_i) t(\pi_i \to \pi_{i+1})\right\} \times e_{k_L}(x_L)$ となります。

例えば、図 3.3 の太い矢印で示した経路を通り、T,A,C,G という配列が観測される確率は、

$e_1(\mathrm{T})t(1 \to 3)e_3(\mathrm{A})t(3 \to 4)e_4(\mathrm{C})t(4 \to 5)e_5(\mathrm{G}) = 0.2 \times 0.4 \times 0.3 \times 0.7 \times 0.4 \times 1.0 \times 0.4 = 0.002688$

となります。確率を掛けていくととても小さな値になるので、しばしば対数が使われます。上記計算に関して自然対数をとると、

$\log(0.2 \times 0.4 \times 0.3 \times 0.7 \times 0.4 \times 1.0 \times 0.4) = \log 0.2 + \log 0.4 + \log 0.3 + \log 0.7 + \log 0.4 + \log 1.0 + \log 0.4 \simeq -5.92$

となります。

3.3 隠れマルコフモデルの構築と未知の配列への適用

隠れマルコフモデルは実際の配列解析にどのように役立てることができるのでしょうか?応用分野の 1 つがシグナル配列やタンパク質のモチーフ配列などの配列パターンの認識です。第 1 章で触れたように、シグナル配列などは一般的に揺らぎを持っています。そこでシグナル配列はどのような塩基配列パターンなのかということを表現するときに、揺らぎを考慮したようなモデルが有効になります。隠れマルコフモデルでは塩基やアミノ酸の揺らぎを考慮し、例えばシグナル配列が AGGA になる可能性は 50%、GGAG になる可能性は 25%のように各々のパターンに対し、確率を割り当てることができます。

このようなモデルを構築し、与えられた配列が出力される確率を求めるには図 3.4 に示すような手順が必要です。まず、隠れマルコフモデルの構造を決定します。具体的にはシグナル配列の特性をうまく表現できるように、どの状態とどの状態を接続するかを決めます。次に、与えられた既知のシグナル配列をもとに、モデル中の状態遷移確率および塩基出力確率を推定します。最後に与えられた未知の配列が、モデルに基づいてどれくらいの確率で出力されるかを計算します。

ここで主に以下の 4 つが問題となります。

1. 隠れマルコフモデルの構造をどうするか？
2. 状態遷移確率、記号出力確率をどう推定するか？
3. 与えられた配列の出力確率をどう求めるか？
4. 与えられた配列の通った経路をどう求めるか？

1 と 2 は隠れマルコフをどのように構築すればよいかという問題になります。3 と 4 は与えられた未知配列の配列に対して隠れマルコフモデルをどのように

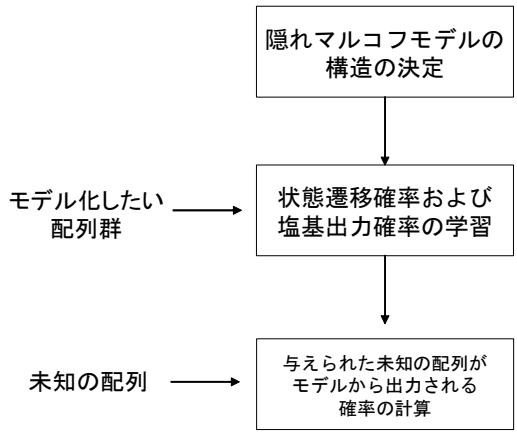


図 3.4　隠れマルコフモデルの構築と未知の配列への適用

適用するかという問題ですが、2 を解く際にも重要な事柄になります。1 については、プロファイル隠れマルコフモデルというものを 3.7 で紹介します。2 は少々難しいので、まずは隠れマルコフモデルの構造と状態遷移確率、記号出力確率が決まっていると仮定し、まず、3 と 4 について考えてゆきましょう。

3.4　最も確率が高い経路の計算

状態 l までにおいて、塩基配列 $x_1, x_2, \cdots, x_i$ が観測されたとします。このとき、どのような状態を経てこのような配列が観測されたのでしょうか？この経路も確率的に決まるため、様々な可能性が考えられます。ここでは、最大の確率を与える経路を求める方法を紹介します。まず最大の確率はいくつになるか、求める方法を考え、この確率を $v(l,i)$ とします。

最大確率を与える経路を通る場合の、状態 l の直前の状態を k_{max} とします (図 3.5)。またこの状態に至る確率を p_{max} とします。すると、

$$v(l,i) = p_{max} t(k_{max} \to l) e_l(x_i) \tag{3.1}$$

となるはずです。

では p_{max} の値はいくつでしょうか？p_{max} は塩基配列 $x_1, x_2, \cdots, x_{i-1}$ が状態 k_{max} に至るまでに観測された確率です。$v(l,i)$ が最大の確率であり、かつ $t(k_{max} \to l)$ と $e_l(x_i)$ が一定である以上、k_{max} に至るまでの確率も最大になっているはずです。したがって、$p_{max} = v(k_{max}, i-1)$ となるはずです。これをもとに式 3.1 を書き直すと、

$$v(l,i) = v(k_{max}, i-1) t(k_{max} \to l) e_l(x_i) \tag{3.2}$$

となります。では k_{max} はどの状態になるのでしょうか？観測 $x_1, x_2, \cdots, x_{i-1}$

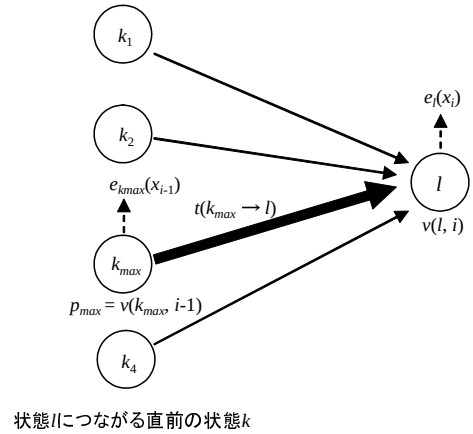


図 3.5　通過した経路が最大確率を与えるときの状態 l の直前の状態

状態 l からは $e_l(x_i)$ の確率で i 番目の塩基 x_i が出力される。この状態に対して、$v(l, i)$ を計算することが可能。状態 l の直前の状態 k をいくつか例として示した。その中には必ず $v(l, i)$ を与える直前の状態 k_{max} およびその時点での確率 p_{max} が存在するはずである。また $p_{max} = v(k_{max}, i - 1)$ が成立する。

がある状態 k に至るまでに観測される最大の確率は $v(k, i-1)$ です。さらにそこから状態 l に遷移して塩基 x_i を出力する確率 P は $v(k, i-1)t(k \to l)e_l(x_i)$ です。k をいろいろ変えてみて、P が最大になるときの (つまり $P = v(l, i)$ となるときの)k の値が k_{max} です。

すると $l = 0$ を開始状態でまだ塩基が何も観測されていない $(i = 0)$ とすると、以下の再帰式が成り立つはずです。

$$v(l,i) = \begin{cases} 0 \text{ if } i = 0 \text{ and } l \neq 0 \\ 1 \text{ if } i = 0 \text{ and } l = 0 \\ e_l(x_i)\max_k(v(k, i-1)t(k \to l)) \end{cases} \tag{3.3}$$

このようにして最大確率を与える経路を求めるアルゴリズムを Viterbi のアルゴリズムと呼びます[11]。

3.5 隠れマルコフモデルの実装

3.5.1 Viterbi のアルゴリズムを使った最大確率の計算

隠れマルコフモデルは状態遷移確率と記号出力確率で決まります。今状態の数が K 個あり、4 種類の塩基を扱うとすると、状態遷移確率は $K \times K$ の行列で表すことができ、また塩基の出力確率は、$K \times 4$ の行列で表すことができます。これを C 言語で表現できるように配列変数を実装します。

3.5.1.1 定数の用意

以下のように、状態の数を表す変数 K、塩基配列の長さを表す変数 L、塩基の種類数を表す変数 NUM_NUC を定義します。塩基配列を扱うときは NUM_NUC を 4 に、アミノ酸配列を扱うときは NUM_NUC を 20 にセットします。

```
#define K 5 /* Number of states */
#define L 10 /* Length of sequence */
#define NUC_NUM 4 /* Number of bases */
```

3.5.1.2 遷移確率の行列

各状態に番号を付与します。下記の例では状態 0 から 4 までが定義されています。二次元配列 t を用いて状態遷移確率を定義します。最初の添字が遷移元の状態、2 番目の添字が遷移先の状態を表します。例えば、t[2][3] は状態 2 から 3 に遷移する確率を表します。

```
static double t[K][K] = {
```

```
/*  0    1    2    3    4  */
  0.0, 0.6, 0.4, 0.0, 0.0, /* 0 */
  0.0, 0.0, 0.0, 1.0, 0.0, /* 1 */
  0.0, 0.7, 0.0, 0.3, 0.0, /* 2 */
  0.0, 0.0, 0.0, 0.3, 0.7, /* 3 */
  0.0, 0.0, 0.0, 0.0, 0.0  /* 4 */
};
```

3.5.1.3 記号の出力確率の行列

各状態における各塩基の出力確率を二次元配列 e を用いて定義します。最初の添字は状態番号を、2 番目の添字は塩基番号を表します。例えば、a,c,g,t をそれぞれ 0,1,2,3 という塩基番号と定義すると、e[2][1] は状態 2 において塩基 c が出力される確率になります。

```
static double e[K][NUC_NUM] = {
/*   a     c     g     t   */
  0.40, 0.30, 0.20, 0.10, /* 0 */
  0.40, 0.10, 0.30, 0.20, /* 1 */
  0.30, 0.20, 0.10, 0.40, /* 2 */
  0.25, 0.25, 0.30, 0.20, /* 3 */
  0.40, 0.25, 0.25, 0.10  /* 4 */
};
```

3.5.1.4 塩基配列

塩基配列を格納する配列変数 x を定義します。本文中では塩基番号は $x_1, x_2, \cdots$ のように 1 番から始まりますが、メモリの有効利用のため、x には 0 番目から塩基を格納します。

```
static char x[L];
```

3.5.1.5 状態のタイプ

各状態のタイプを配列 type 定義します。TYPE_S は状態遷移系列の最初に該当する開始状態、TYPE_N は通常の塩基を出力する状態、TYPE_D は塩基の欠損を表す状態を表します (後述)。TYPE_S と TYPE_D は塩基を出力しません。添字は状態番号を表し、type[2] = TYPE_N は 2 番目の状態が通常の塩基を出力

する状態であることを表します。

TYPE_S、TYPE_N、TYPE_D などには#define を使って値の異なる定数を割り当てます。

```
static int type[K] = { TYPE_S, TYPE_N, TYPE_N, TYPE_N, TYPE_N };
```

3.5.1.6 Viterbi のアルゴリズムの実装

viterbi のアルゴリズムをプログラムとして実装すると以下のようになります。viterbi(l, i) で状態 l において i 番目の塩基が観測される最大の確率が返ってきます。

```
#define LOG0 -1000.0 /* 無限小の意味で LOG0 を定義 */

/* 状態 l において i 番目の塩基が観測される最大の確率を計算する
   確率の log が返ってくる */
double viterbi(int l, int i){

  int k, k_max;
  double p, p_max;

  switch(type[l]){ /* 状態 l のタイプを判断 */
  case TYPE_S:
    if(i <= 0)return 0.0;
    /* 開始状態で塩基が出力されていないなら、確率=1(対数をとると、0) */
    /* これは、式 3.3 の右辺の 2 番目の式に該当する */

    else return LOG0;
    /* 開始状態なのに塩基が出力されているなら、
       確率=0 (対数をとると、−∞) */
    /* これは、式 3.3 の右辺の最初の式に該当する */

  case TYPE_N:
    if(i <= 0)return LOG0;
    for(p_max = LOG0 - 1, k = 0;k < K;k ++){
    /* 最大の確率を与える状態 k_max およびその確率 p_max を探す */
    /* これは、式 3.3 の右辺の 3 番目の式に該当する */
```

```
      if(t[k][l] > 0.0)p = viterbi(k, i - 1) + log(t[k][l]);
      else p = LOG0;
      if(p > p_max){ p_max = p; k_max = k; }
    }
    return p_max + log(e[l][ (int)cton(x[i - 1]) ]);
       /* cton は塩基 a,c,g,t を 0,1,2,3 に変換する関数。
          これは自分で作りましょう。 */
  }
}
```

3.5.2 計算の重複の回避

各状態 l および i 番目の塩基に対して、$v(l,i)$ の計算をすることが可能です。$v(l,i)$ を呼び出すと、状態 l へつながっている各状態 $k_1, k_2 \cdots$ に対して $v(k_1, i-1), v(k_2, i-1), \cdots$ が呼び出されます。関数 v を呼び出すと、このように l,i に様々な値が入って v が再帰的に呼び出されます。同じ l,i の値に対して $v(l,i)$ の計算が複数回計算が行われることも多々あり、その分だけ計算時間が無駄になります。

そこで配列 double v[l][i] を用意し、計算の途中経過を v[l][i] に記録することにより高速化が可能です (課題 3.3)。基本的には A.3 で学習した方法と一緒で、最初の呼び出しでは、計算を行ってその結果を v[l][i] に記録します。2 度目からは、v[l][i] に記録した値を返すだけにします。

3.5.3 経路を求める

ここまで Viterbi のアルゴリズムを使って最大の確率を計算する手法を学びました。しかし毎回関数 viterbi を呼び出すときに得られるのは、最大の確率とその確率を与える**直前**の状態だけです。そこで最大の確率を与える経路はどのような状態を通るかを求める方法を実装という面から紹介します (演習問題 3.4。

まず配列 int ptr[][] を用意します。ptr[l][i] には状態 l で x_1〜x_i が観測されるときの最適の経路の直前の状態を記録するのです。具体的には関数 viterbi の適切な場所で、

```
ptr[l][i] = k_max;
```

を入れ、下のような関数 ptr_to_path を作ります。

```
int ptr_to_path(int l, int i){
```

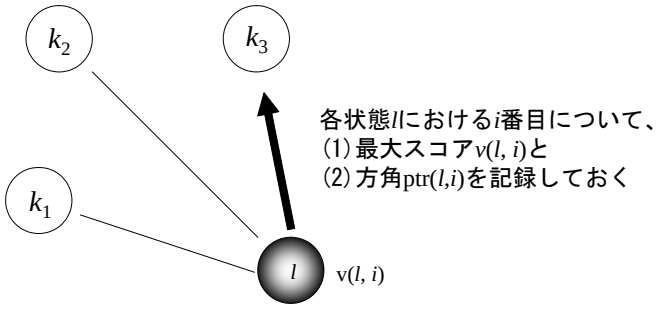
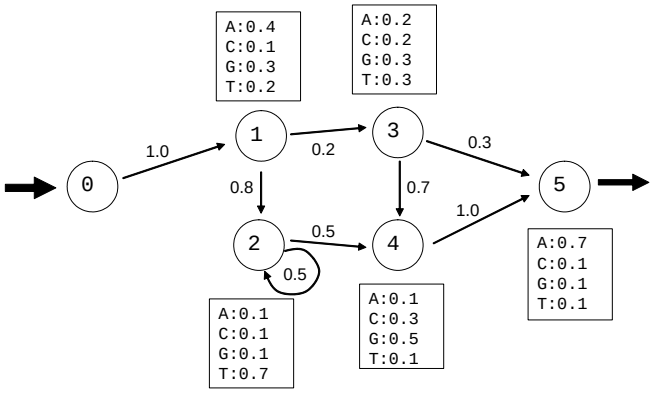


図 3.6　通過した経路が最大確率を与えるときの直前の状態 l とその確率の記録

```
  int n = 0;
  int new_l;

  path[n ++] = l;
  while(type[l] != TYPE_S){
    new_l = ptr[l][i];
    if(type[l] == TYPE_N)i --;
    l = new_l;
    path[n ++] = l;
  }

/* path の中身を逆順にする処理をここに加えてもよい */

  return n;

}
```

この関数では、各状態の直前の状態を次々と開始状態まで辿っていくことにより、経路を配列 path に順次記録してゆきます。最終的には path には通ってきた状態が全て入ることになります。

結局、各ノードには図 3.6 に示すように、最大確率 $v(l,i)$ の情報と、その最大確率に至るときの直前のノードの情報 $\mathrm{ptr}(l,i)$ を持たせれば、高速な計算を行いつつ、通ってきた経路を辿ることができるようになります。図 3.7 に $v(l,i)$ と $\mathrm{ptr}(l,i)$ の中味の例を示しておきましょう。



$v(l, i)$ 配列のi番目

状態l	0	1	2	3	4
0	1				
1		0.4			
2			0.224		
3			0.024		
4				0.056	
5					0.0392

ptr(l, i) 配列のi番目

状態l	0	1	2	3	4
0					
1		0			
2			1		
3			1		
4				2	
5					4

経路：５←４←２←１←０

図 3.7 ”ATGA”を Viterbi のアルゴリズムにかけたときの $v(l,i)$, $\mathrm{ptr}(l,i)$ の中味
一番上に隠れマルコフモデルの例を、中段に各状態における $v(l,i)$、下段に各状態における $\mathrm{ptr}(l,i)$ を示した。中段で値が 0 になるところは空欄にした。下段の表より、通過した状態を辿ることができる。

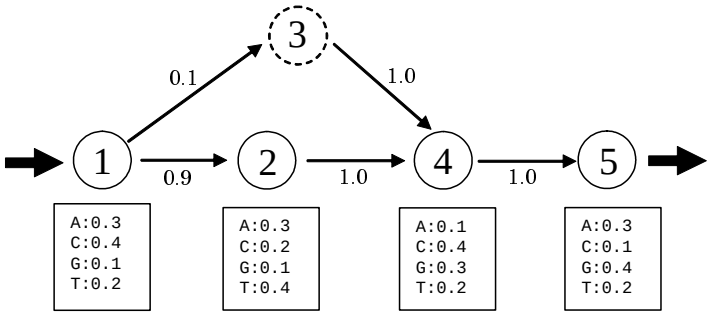


図 3.8　欠損を表す状態

これは通常 4 塩基を出力する隠れマルコフモデルである。状態 3 が塩基の欠損を表す。下の経路を通ると 4 塩基が出力されるが、上の経路を通ると状態 2 の塩基が欠損し、3 塩基となる。

3.6　欠損を表すノード

記号を出力しない状態を作ると、その状態は塩基の欠損を表します。例えば本来 4 塩基の配列パターンからなるシグナル配列の 1 塩基が欠損したというケースを表現したいときは、4 つの普通の状態と、1 つの欠損を表す状態で表現することができます (図 3.8)。3.5.1.6 に掲載したプログラムの場合、`TYPE_D` をそれに該当させます (演習問題 3.5)。

欠損を表すノードに関しての Viterbi の再帰式は、

$$v(l,i) = \max_k (v(k,i)t(k \to l)) \tag{3.4}$$

となります。

なお欠損を表す状態の場合、自分自身に遷移するような確率 $t(l \to l)$ は 0 にしておきます。

3.7　プロファイル隠れマルコフモデル

隠れマルコフモデルをどのような構造にすれば、効果的にシグナル配列をモデル化できるでしょうか？これはモデル化する配列の性質によるので、一概に論じることはできません。しかし汎用的に使うことが可能な構造の例として、プロファイル隠れマルコフモデルがあります [11]。これは図 3.9 に示すような構造になっています。まず下に通常の塩基を出力する状態の列が並んでいます。これはシグナル配列のコンセンサス配列に該当します。一番下の経路を通ると、揺らぎを含んだシグナル配列が出力されることになります。中段の経路は塩基

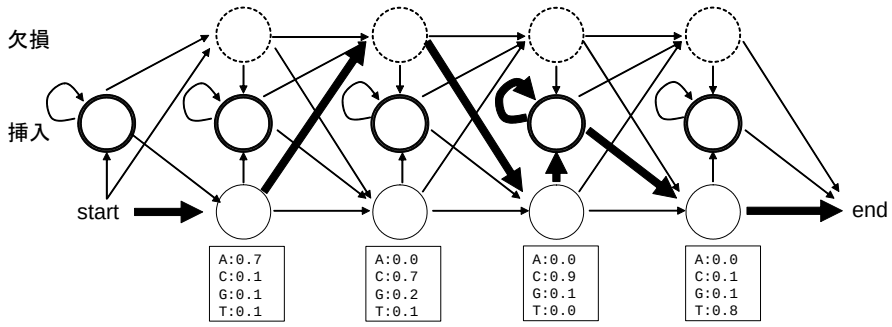


図 3.9　プロファイル隠れマルコフモデル

一番下の状態の列が、塩基を出力する通常の状態で、シグナル配列のコンセンサス配列に該当する。中段の状態の列が塩基の挿入を表す。上段の状態の列は塩基の欠損を表す。本文中で議論している経路を太い矢印で示す。状態遷移確率、および中段の塩基出力確率は省略した。

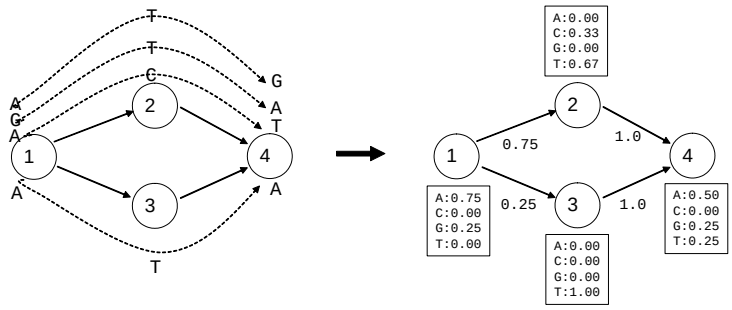


図 3.10　簡単な隠れマルコフモデルの学習

左の図の点線は、Viterbi のアルゴリズムで点線に重なる配列に対して求められた最大の確率を与える経路である。この経路を元に遷移確率および塩基出力確率を推定すると右のようになる。例えば 1 → 2 は 3 回、1 → 3 は 1 回使われているので、1 → 2 の遷移が起こる確率は 3/4=0.75 となる。また 2 の状態では T が 2 回、C が 1 回出力されているので、状態 2 で T が出力される確率は 2/3=0.67 となる。

の挿入を表します。この状態に入ると 1 個またはそれ以上の塩基が挿入されます。一番上は、塩基の欠損を表します。この状態は塩基を出力しません。

例えば、図 3.9 の場合、最も出力確率が高い塩基を見ればコンセンサス配列は ACCT となります。太い矢印で示したような経路を通って塩基が出力された場合、2 塩基目が欠損し、3 塩基目のあとに 2 塩基以上の塩基の挿入が起こり、最後にコンセンサス配列の 4 番目の塩基が出力されます。すると例えば、ACggT のような配列パターンが出力されます。ここで小文字は塩基の挿入を表します。

3.8　簡単なパラメータの学習

それでは、$e_l(b)$ や、$t(k \to l)$ などのパラメータはどのように決めればいいのでしょうか？基本的にはモデリングしたい配列をなるべく多く用意し、それをもとにパラメータを決めることになります。例えば、ヘリックス・ループ・ヘリックスのタンパク質のモチーフを隠れマルコフモデルでモデリングする場合、ヘリックス・ループ・ヘリックスのモチーフを含むようなタンパク質の配列を沢山用意します。次に各配列がモデル上のどの経路を通ったか分かっている場合は、通った経路、および通った状態から出力された塩基を各経路、各状態ごとに集計します。そしてその集計結果をもとに、遷移確率および塩基の出力確率を推定します。しかし通常は通った経路は不明です。そこで Viterbi のアルゴリズムを使って、最も通った確率が高くなる経路を求め、それを集計します (図 3.10)。

今 n 本の配列が与えられているとして、上記をより形式的に表現します。

1. まず、$e_l(b)$ や、$t(k \rightarrow l)$ の値をランダムに割り振ります。但し、$\sum_{b'} e_l(b') = 1, \sum_{l'} t(k \rightarrow l') = 1$ となるようにします。ここで b' は任意の塩基 (a,c,g,t)、l' は k から 1 回で到達できる任意の状態（次の状態）を表します。
2. n 本の配列について Viterbi のアルゴリズムを使って最適経路を求めます。このとき、使われた塩基と経路を記録しておきます。
3. そして以下の式により全ての $e_l(b)$、$t(k \rightarrow l)$ を更新します。

$$e_l(b) = \frac{n(l,b)}{\sum_{b'} n(l,b')} \tag{3.5}$$

$$t(k \rightarrow l) = \frac{n(k \rightarrow l)}{\sum_{l'} n(k \rightarrow l')} \tag{3.6}$$

   但し、$n(l,b)$ は状態 l において塩基 b が使われた回数、$n(k \rightarrow l)$ は状態 k から状態 l へ移る経路が使われた回数を表します。
4. $e_l(b)$ や、$t(k \rightarrow l)$ の値が収束する（変化しなくなる）まで 2 に戻って繰り返します。

3.9 配列が観測される確率の計算

3.4 では、塩基の観測確率および状態間の遷移確率が決まっているときに与えられた配列 $x = (x_1, x_2, \cdots, x_L)$ が通った経路 π から計算される確率が最大になるような経路およびその確率を求める手法 Viterbi のアルゴリズムを解説しました。この手法はより形式的に以下のように表現することができます。

状態の集合 $\Upsilon = \{k_1, k_2, \cdots, k_K\}$ および塩基の集合 $B = \{\mathrm{a}, \mathrm{c}, \mathrm{g}, \mathrm{t}\}$ に対して、隠れマルコフモデルを決定する上でのパラメータとなる、塩基の観測確率と状態間の遷移確率 $\theta = (e_k(b), k), k \in \Upsilon, b \in B$ が決まっているとします。Viterbi のアルゴリズムは与えられたパラメータ θ のもとで、配列 x が観測される確率が最大になるような経路 π=argmax$_\pi P(x, \pi|\theta)$ およびその確率 $\max_\pi P(x, \pi|\theta)$ を求めるアルゴリズムであると言えます。ここからさらに一歩進んで、最大確率だけでなく、与えられた配列 x が隠れマルコフモデルから出力される確率 $P(x|\theta)$ を求める方法を考えましょう。一番簡単な方法は配列 x が出力されるときに採りうる経路 π を全て列挙し、以下のようにその経路を通るときの確率を全て合計する方法です。

$$P(x|\theta) = \sum_{\pi} P(x, \pi|\theta) \tag{3.7}$$

しかしこの方法では状態の数が多くなると、計算すべき経路の数も爆発的に増える可能性があります。そこでこれを効果的に計算する**前向きアルゴリズム**

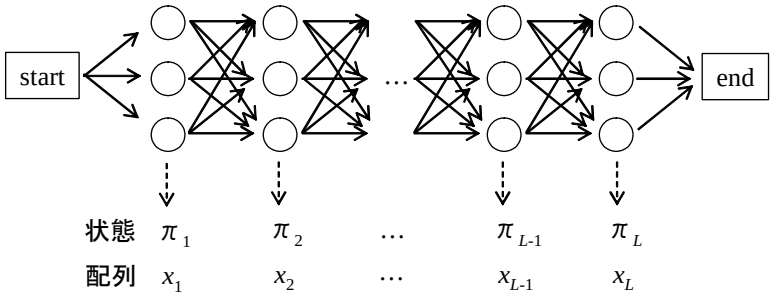


図 3.11　単純な隠れマルコフモデルの構造

経路の最初は start から始まり、end で終わる。各状態で必ず塩基が出力される。すなわち、経路上の状態の数と、配列に含まれる塩基の数が等しくなる。

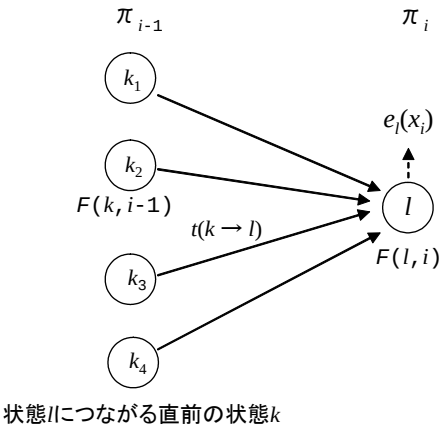


図 3.12　前向きアルゴリズム

$F(l,i)$ は状態 l の経路までにおいて、塩基 $x_1,\cdots,x_i$ が観測される確率を表す。状態 l の直前の状態 k から l に至り、x_i が出力される確率は、$F(k,i-1)t(k\to l)e_l(x_i)$ となる。

および**後向きアルゴリズム**を紹介しましょう [11]。

まず説明を分かりやすくするために、図 3.11 のように start から始まって、end で終わるような単純な隠れマルコフモデルの構造を考えます (3.4 では start は状態 0 として扱われていました)。そして、欠損状態はないと仮定します (つまり start から end に至る経路上途中で通過した状態の数と塩基配列の長さが等しいとします)。

前向きアルゴリズムでは、以下の式 3.8 のように状態 k に到り、かつそのときに配列 $x_1,x_2,\cdots,x_i$ が観測されている確率 $F(k,i)$ を求めます。

$$F(k,i)=P(x_1,\cdots,x_i,\pi_i=k) \tag{3.8}$$

F を使うと、$P(x)$ は式 3.7 より、

$$P(x|\theta)=\sum_k P(x_1,\cdots,x_L,\pi_L=k|\theta)t(k\to\mathrm{end})$$

$$=\sum_k F(k,L)t(k\to\mathrm{end})$$

となります。

それでは F を効率よく求める方法を考えましょう。まず、配列が何も観測されていなければ $(i=0)$、必ず初期状態 start です。従って、$F(\mathrm{start},0)=1$ となります。それ以外の場合については、図 3.12 を見ながら $F(l,i)$ を求めることを考えてゆきます。l に到る直前の状態 π_{i-1} として、$k\in\Upsilon$ を考えます。こ

れらの k に対してさらに $F(k, i-1)$ を考えることができ、k から l への遷移確率は $t(k \to l)$ となります。従って $F(k, i-1)t(k \to l)$ は状態 k までに配列 $x_1, \cdots, x_{i-1}$ が観測され、そこから状態 l に遷移する確率となります。これに l から記号 x_i が出力される確率 $e_l(x_i)$ を掛ければ、状態 $k \to l$ の遷移が起こった段階で配列 $x_1, \cdots, x_i$ が観測される確率となります。これを全ての k について計算して総和をとると、

$$F(l, i) = e_l(x_i) \sum_k F(k, i-1)t(k \to l)$$

となります。結局ここまでの議論を合わせると、以下の再帰式をたてることができます。

$$F(l, i) = \begin{cases} 0 \text{ if } l \neq \text{start and } i = 0 \\ 1 \text{ if } l = \text{start and } i = 0 \\ e_l(x_i) \sum_k F(k, i-1)t(k \to l) \text{ otherwise} \end{cases} \tag{3.9}$$

一方後向きアルゴリズムでは、i 番目の状態として k を通過したときに、配列 $x_{i+1}, \cdots, x_L$ が観測される確率 $B(k, i)$ を求めます。これを式で書くと、

$$B(k, i) = P(x_{i+1}, \cdots, x_L | \pi_i = k) \tag{3.10}$$

となります。すると、

$$P(x) = P(x_1, \cdots, x_L | \pi_0 = \text{start}) = B(\text{start}, 0)$$

となります。

では同様に B を効率よく求める方法を考えましょう。記号が全て出尽くした状態 (π_L) では、もう塩基が出力されませんので、あとは終了状態への遷移確率だけが問題となり、$B(k, L) = t(k \to \text{end})$ となります。それ以外の場合は、現在の状態 k の直後の状態 $l \in \Upsilon$ を考えます (図 3.13)。これらの状態 l に対しても $B(l, i+1)$ を考えることができ、k から l への遷移確率は $t(k \to l)$ です。従って、$t(k \to l)B(l, i+1)$ は状態 k から l に遷移し、そこからさらにどこかに遷移して配列 $x_{i+2}, \cdots, x_L$ が出力される確率となります。これに状態 l から塩基 x_{i+1} が出力される確率 $e_l(x_{i+1})$ を掛ければ状態 k から l への遷移が起こり、その後配列 $x_{i+1}, \cdots, x_L$ が観測される確率となります。これを全ての l について計算して総和をとると、

$$B(k, i) = \sum_l t(k \to l)e_l(x_{i+1})B(l, i+1)$$

となります。以上の議論をまとめると、以下のような再帰式をたてることができます。

$$B(k, i) = \begin{cases} t(k \to \text{end}) \text{ if } i = L \\ \sum_l t(k \to l)e_l(x_{i+1})B(l, i+1) \end{cases} \tag{3.11}$$

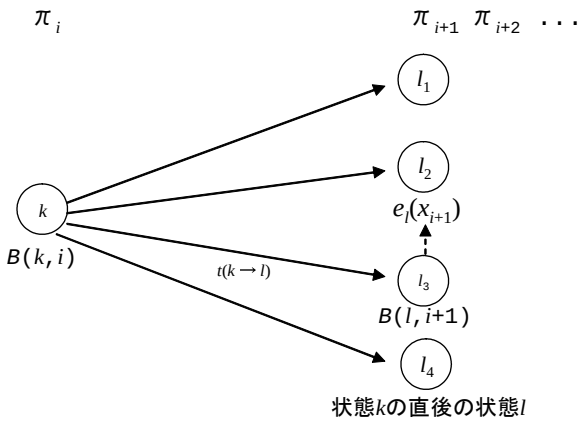


図 3.13　後向きアルゴリズム

$B(k,i)$ は i 番目の塩基が状態 k で観測されているとき、それ以降の経路で塩基配列 $x_{i+1},\cdots,x_L$ が観測される確率である。状態 k から状態 l に遷移し、塩基 x_{i+1} が観測される確率は、$t(k\to l)e_l(x_{i+1})B(l,i+1)$ となる。

3.10 隠れマルコフモデルのパラメータ推定

それでは複数の配列が与えられたときに、隠れマルコフモデルのパラメータをどのように推定していけばよいか、最尤推定量を使った方法を紹介します。最尤推定量は与えられた観測データ x に対して、$P(x|\theta)$ を最大にするような θ のことであり、パラメータ推定の方法として広く使われています。これを数値的に決める方法について説明していきます (理論的背景については、C.3 参照)。

配列 x と隠れマルコフモデルのパラメータ θ が与えられたときに、状態 k、l がそれぞれ i 番目の経路および $i+1$ 番目の経路として使用される確率は

$$
\begin{aligned}
&P(\pi_i = k, \pi_{i+1} = l|x, \theta) = \frac{P(x, \pi_i = k, \pi_{i+1} = l|\theta)}{P(x|\theta)} \\
&= \frac{P(x_1, \cdots, x_i, \pi_i = k|\theta) P(\pi_i = l|\theta) P(x_{i+1}|\pi_{i+1} = l|\theta)}{P(x|\theta)} \\
&\times P(x_{i+2}, \cdots, x_L|\pi_{i+1} = l, \theta) \\
&= \frac{F(k,i)_{|\theta} t(k \to l)_{|\theta} e_l(x_i + 1)_{|\theta} B(l, i+1)_{|\theta}}{P(x|\theta)} \qquad (3.12)
\end{aligned}
$$

となります。但し、$F_{|\theta}$、$B_{|\theta}$ はその計算がパラメータ θ をもとに行われることを示し、$t(k \to l)_{|\theta}$ や $e_k(b)_{|\theta}$ などはパラメータ θ の一部であることを表します。

経路 π 上の任意の位置 $i, i'(i \neq i')$ について、i 番目の状態から $i+1$ 番目の状態に遷移するときに $k \to l$ の経路が使われる確率と、i' 番目の状態から $i'+1$ 番目の状態に遷移するときに $k \to l$ の経路が使われる確率が独立なら、何番目の経路かということに関係なく、経路 $k \to l$ が配列 x を出力する上で使われる回数の期待値は式 3.12 を塩基間の全ての状態遷移の箇所について足し合わせ[*1)]、

$$
\begin{aligned}
&\sum_i P(\pi_i = k, \pi_{i+1} = l|x, \theta) \\
&= \frac{1}{P(x|\theta)} \sum_i F(k,i)_{|\theta} t(k \to l)_{|\theta} e_l(x_i + 1)_{|\theta} B(l, i+1)_{|\theta} \qquad (3.13)
\end{aligned}
$$

と表されます。すると複数本の配列 $x^1, x^2, \cdots$ が与えられたとき、経路 $k \to l$ が使われる回数の期待値 $N_{t(k \to l)}$ は式 3.13 をそれらの配列 x^j について足し合わせ、

---

*1) 一般に確率変数 $X_1, X_2, \cdots, X_n$ が独立な場合、$\sum_{k=1}^n X_k$ の期待値 $E(\sum_{k=1}^n X_k)$ は $\sum_{k=1}^n E(X_k)$ となる。例えば"5" の目が出る確率が 1/6 のサイコロを 1 回ふった場合、"5" の目が 1 回出るか 0 回かのどちらかである。この回数を X で表す。k 回目にサイコロを**1回**ふったとき、"5" が出た回数 (0 または 1) を X_k で表すとして、その期待値は $E(X_k) = 1 \times \frac{1}{6} + 0 \times \frac{1}{6} = \frac{1}{6}$ となる。各試行は独立なので、30 回ふったときに"5" の目が出る回数は、$E(\sum_{k=1}^{30} X_k) = \sum_{k=1}^{30} E(X_k) = \frac{1}{6} \times 30 = 5$ となる。

$$N_{t(k\to l)|\theta} = \sum_j \frac{1}{P(x^j|\theta)} \sum_i F^j(k,i)_{|\theta} t(k\to l)_{|\theta} e_l(x^j_{i+1})_{|\theta} B^j(l,i{+}1)_{|\theta} \quad (3.14)$$

となります。

一方、i 番目の塩基が $b(x_i = b)$ として、i 番目の状態 k が塩基 b の出力に使われる確率 $P(\pi_i = k|x,\theta)$ は

$$\begin{aligned} P(\pi_i = k|x,\theta) &= \frac{P(x,\pi_i = k|\theta)}{P(x|\theta)} \\ &= P(x_1,\cdots,x_i,\pi_i = k|\theta) P(x_{i+1},\cdots,x_L|x_1,\cdots,x_i,\pi_i = k,\theta) \\ &= \frac{P(x_1,\cdots,x_i,\pi_i = k|\theta) P(x_{i+1},\cdots,x_L|\pi_i = k,\theta)}{P(x|\theta)} \\ &= \frac{F(k,i)_{|\theta} B(k,i)_{|\theta}}{P(x|\theta)} \end{aligned} \quad (3.15)$$

となります。$x_i = x_{i'} = b$ を満たす経路 π 上の任意の位置 i, i' について i 番目の状態 k が塩基 b の出力に使われる確率と、i' 番目の状態 k が塩基 b の出力に使われる確率が独立の場合、状態 k において記号 b が出力される回数の期待値は、式 3.15 を $x_i = b$ となる i について足し合わせて

$$\frac{1}{P(x|\theta)} \sum_{\{i|x_i=b\}} F(k,i)_{|\theta} B(k,i)_{|\theta} \quad (3.16)$$

となります。式 3.16 を複数本の配列 $x^1, x^2, \cdots$ が与えられたときの期待値に直すと、

$$N_{e_k(b)|\theta} = \sum_j \frac{1}{P(x^j|\theta)} \sum_{\{i|x^j_i=b\}} F^j(k,i)_{|\theta} B^j(k,i)_{|\theta} \quad (3.17)$$

となります。

さて、式 3.14、3.17 を用いて以下の手順でパラメータ θ を推定してゆきます。まず θ の初期値を決めます。この決め方にはいくつか方法がありますが、とりあえずここではランダムに決めることにしましょう。真の θ と、とりあえず決めた θ を区別するために、ある時点 t における後者を θ^t とします。そして現在の θ^t に基づいて、式 3.14、3.17 を用いて、$N_{t(k\to l)|\theta^t}$ および $N_{e_k(b)|\theta^t}$ を計算します。次に θ^{t+1} を以下の式により更新します。

$$t(k\to l)_{|\theta^{t+1}} = \frac{N_{t(k\to l)|\theta^t}}{\sum_{l'} N_{t(k\to l')|\theta^t}} \quad (3.18)$$

$$e_k(b)_{|\theta^{t+1}} = \frac{N_{e_k(b)|\theta^t}}{\sum_{b'} N_{e_k(b')|\theta^t}} \quad (3.19)$$

次にまた $N_{e_k(b)|\theta^{t+1}}$ および $N_{t(k\to l)|\theta^{t+1}}$ を計算する、ということを繰り返すうちに、θ^t は $P(x|\theta)$ を最大化するような θ に近づいてゆきます。

演習問題

3.1　logAB = logA + logB を使って式 3.3 を log を使うように直しましょう。ただし log0 = LOG0 とします。

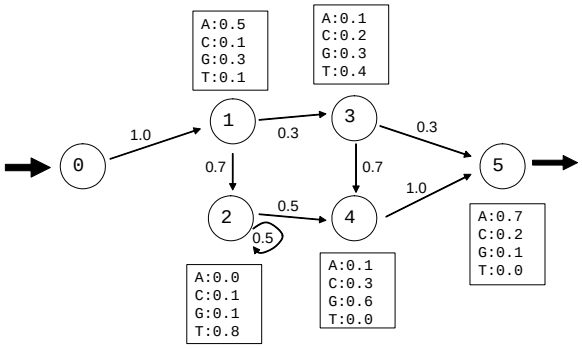
3.2　3.5.1.6 のプログラムを参考に状態 l において塩基配列 x_1〜x_i が観測されるときの最大の確率を計算する関数 viterbi を完成させましょう。

3.3　計算の途中経過を v[l][i] に記録することにより 3.2 を高速化しましょう。

3.4　3.5.3 のプログラムを参考に、最も確率が高くなる経路を求めるプログラムを作成しましょう。

3.5　欠損を表すノードをプログラムとして実装しましょう。

3.6　図 3.7 を参考に下記隠れマルコフモデルについて、”ATGA”が出力されたときの $v(l,i)$, $\mathrm{ptr}(l,i)$ を求め、表にしましょう。



3.7　Viterbi のアルゴリズムを使って遷移確率および塩基出力確率の学習アルゴリズムを実装しましょう。

第 4 章
RNA の二次構造予測

RNA 分子は A,C,G,U の塩基からなる一本鎖の核酸配列です。二本鎖である DNA とは異なり、比較的自由に折れ曲がることができるため、その立体構造も様々です。RNA には mRNA や tRNA,rRNA など様々な種類のものがあります。また最近では生体内でタンパク質に翻訳されない RNA がたくさんあることが判明し、注目されています。RNA 分子の機能は多くの場合、その立体構造に依存しています。従って RNA の立体構造と機能との関係を調べることは大変重要です。RNA の三次元構造を配列から予測することはまだ困難ですが、単にどの塩基とどの塩基が対合するかということを予測する二次構造予測に関しては手法がある程度確立されています。そこで本章では、与えられた RNA 分子がどのような二次構造をとるかを予測する手法について解説します。

4.1 RNA とその二次構造

RNA 分子は DNA と同様、細胞内に多く存在する核酸の一種です。RNA にはタンパク質を合成するときの鋳型となる mRNA、コドンをアミノ酸に変換する tRNA、リボソームの中に存在してタンパク質の合成を行う rRNA など様々な種類のものが存在します。また最近では細胞内にはタンパク質に翻訳されない非翻訳 RNA が多数存在し、その一部は生体内で重要な役割を果たしていることが分かってきました [12]。

DNA は二本鎖であり、あまり自由に折れ曲がることができないため、mRNA の鋳型になる以外の機能は限られています。これに対し、RNA は一本鎖であり、DNA に比べると比較的自由に折れ曲がることができるため、その構造によって様々な機能を持つことが可能なのです。従って RNA の機能を考える上で、その構造を調べることは非常に重要です。

RNA は A,C,G,U の 4 文字からなる配列で、A と U、C と G は結合しやすくなっています。RNA 配列の中でどの塩基とどの塩基が結合しているかを表

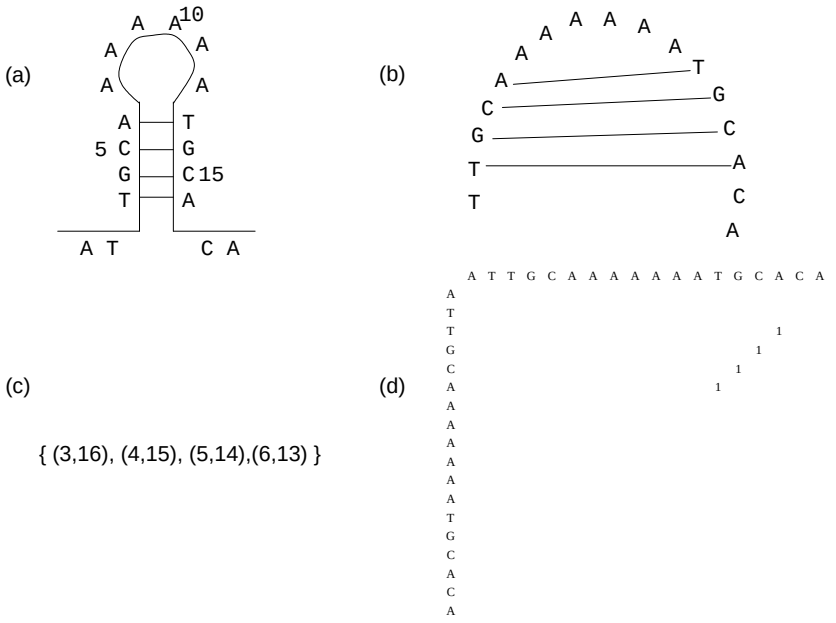


図 4.1　RNA 二次構造の表現方法

現したものを RNA の二次構造と呼びます。RNA の構造を決める大きな要因はその一次配列です。一次配列から三次元構造を予測することが現段階ではまだ困難ですが、二次構造を予測する手法は実用化されています [13]。そこで本章では、RNA の二次構造を予測する方法について学びましょう。

4.2　RNA 二次構造の表現

まず、RNA の二次構造を扱うためには、それをどのように表現するかを考えなければなりません。RNA の二次構造を表現する最も直接的な方法は、図 4.1(a) のように実際に図を描くことです。また、配列を円形にして対合部分を線で結んだのが図 4.1(b) です。これらは直感的には分かりやすいのですが、二次構造を数学的に扱ったり、プログラムとして実装したりすることを考えるときは必ずしも直接扱いやすい表現方法とは言えません。そこでまず RNA 配列の各塩基に対して、先頭より順番に番号を付け、図 4.1(c) のように、対合している組を列挙する方法が考えられます。これは数学では集合として扱うことができます。そして図 4.1(d) では配列を縦横に並べ、対合している部分に”1”を記入しています。これも厳密な表現方法であり、コンピュータでも処理しやすいと言えます。本章では RNA 二次構造に関して図を用いて直感的に解説するときは主に図 4.1(a) または (b) の表現方法を、RNA 二次構造を数学的あるいはコンピュータで扱うことを考えるとき、図 4.1(c) または (d) の表現方法を使

用することにします。

4.3 RNA 二次構造の評価

では最初に RNA 二次構造を行う簡単な方法から考えましょう。まず良い二次構造とは何かを決めなければなりません。そこでまず、ある二次構造に対してスコアを決めるような枠組みを定義します。このとき、そのスコアが低ければ低いほど、実際に RNA がそのスコアに対応する構造をとっている可能性が高いようにします。このようなスコアとして自由エネルギー*1)がよく使われます。厳密な議論はここでは避けますが、自由エネルギーは状態の安定性を表す指標で、自然界ではこの数値が小さくなる方向へ状態が自発的に変化します [14]。従って、この数値が小さければ小さいほどその状態は安定になります。RNA 配列中のどの塩基も対合していないような状態の自由エネルギーを 0 とすると、RNA がとりうる二次構造に対応する自由エネルギーは当然負の数になります。

無数に考えられる RNA の構造 1 つ 1 つに対して自由エネルギーを実験的に調べるのは事実上不可能なので、例えば A-U 結合の自由エネルギーは -2.0、C-G 結合の自由エネルギーは -3.0 のように各部分の塩基対や構造に対して自由エネルギーを定義し、その総和を構造全体の自由エネルギーとする近似計算がよく行われます。そこでこの計算法を数学的に以下のように表現します。

まず、RNA 分子を n 文字からなる文字列 $R = r_1 r_2 r_3 \cdots r_n (r_i \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{U}\})$ と考えます。RNA の二次構造は図 4.1(c) に既に示したように、(r_i, r_j) の対の集合 S です。ここで i と j は $1 \leq i < j \leq n$ を満たします。$(r_i, r_j) \in S$ なら原則として r_i と r_j は A-U, C-G などの相補的な塩基対でなければなりません*2)。

そして RNA 中のある場所の 2 つの塩基が対合したときの自由エネルギーは他の場所の構造に関係ないと仮定します。この仮定は RNA の二次構造予測を容易にする上で重要です。さてこの仮定のもとでは、構造 S の自由エネルギーの合計 E は、

$$E(S) = \sum_{(r_i, r_j) \in S} \alpha(r_i, r_j) \tag{4.1}$$

となります。ただし、$\alpha(r_i, r_j)$ は (r_i, r_j) の対合による自由エネルギーです。

実際には RNA はあまりきつく曲がることができないので、ある特定の閾値 t に対し、$j - i > t$ でなければなりません。そこでそのような構造に対しては大きな自由エネルギーを割り当てれば、このような構造を回避することができます。

---

*1) ギブスの自由エネルギー＝エンタルピー－温度×エントロピー。

*2) 自然界には G-U の弱い結合が存在する。

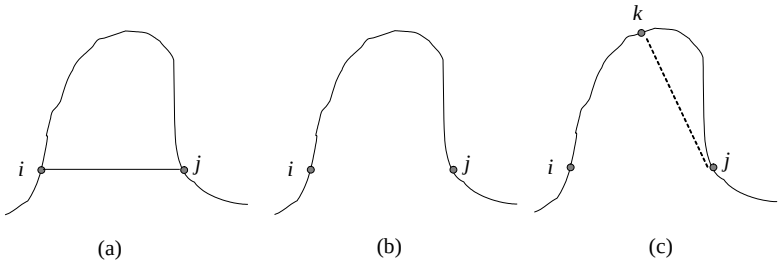


図 4.2　部分構造 $S_{i,j}$ の 3' 末端の塩基 r_j が取りうる 3 つの状態

これで RNA の二次構造に対して、自由エネルギーというスコア体系を定義することができました。

4.4　簡単な RNA 二次構造予測

RNA 二次構造予測法について触れる前に、自然界にある二次構造で対象から外さなければならないものがあることをまず説明しておきましょう。それはノット構造です [13]。ノット構造は $(r_i, r_j) \in S, \quad (r_k, r_l) \in S$ でかつ $i < k < j < l$ を満たすときに存在する構造です。このような構造は確かに自然界に存在し、その一部は重要な機能を持っていますが、これを除外すれば二次構造予測の問題が簡単になるのです*3)。

さていよいよ RNA の二次構造予測に対してどのようなアルゴリズムが考えられるか、考えてゆきましょう。最適な二次構造は最小の自由エネルギーを持つような構造です。従って、二次構造予測は最小の自由エネルギーをもつ構造を探し出すことに帰着します。そこで最も単純なアルゴリズムは、可能な構造を全て列挙し、その中から自由エネルギーが最も低いものを選ぶものです。しかし可能な構造の数はその塩基数に対して指数関数的に増大するので、長い RNA に対してこのアルゴリズムを適用するのは計算時間という意味で実用的ではありません。

そこでまず部分配列 $R_{i,j} = r_i r_{i+1} \cdots r_j$ が取りうる二次構造 $S_{i,j}$ の中で最小の自由エネルギーを持つものを効率よく探す方法を考えましょう。ここで $R_{i,j}$ の 3' 末端の塩基 r_j の状態として考えられるのは、図 4.2 に示すように、(a)r_i と結合しているか、(b) 他の塩基とは結合していないか、(c)r_i 以外の塩基と結合しているかの 3 通りです。

もし r_j が r_i と対合しているとすれば、この結合による自由エネルギーは、$\alpha(r_i, r_j)$ となります。ここで、RNA 中のある場所の 2 つの塩基が対合したときの自由エネルギーは他の場所の構造には影響されないという仮定を思い出し

*3)　ノットがなければ S は平面グラフになる。

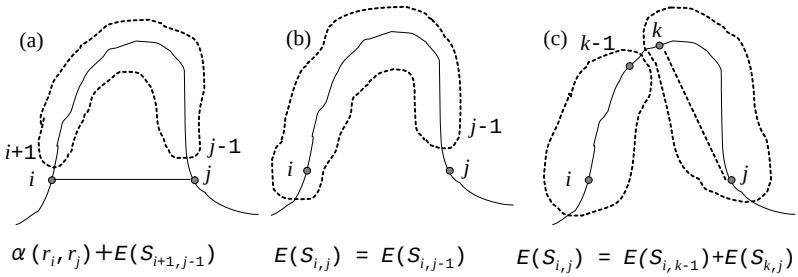


図 4.3 RNA の最適な二次構造を求めるための 3 つの再帰関係

て下さい。つまり、r_j が r_i と対合しているときの自由エネルギーはその内側の構造に関係なく常に $\alpha(r_i, r_j)$ で一定です。従って、r_j と r_i が対合している状態で構造全体の自由エネルギーを最小にするためには、$S_{i+1,j-1}$ の部分のエネルギーを最小にすればよいことになります。これを式で書くと、

$$E(S_{i,j}) = \alpha(r_i, r_j) + E(S_{i+1,j-1}) \tag{4.2}$$

となります[*4](図 4.3(a))。

もし r_j の塩基が他の塩基と対合していないとすれば、r_j の塩基は自由エネルギーの計算には関与しないことになるので、

$$E(S_{i,j}) = E(S_{i,j-1}) \tag{4.3}$$

となります (図 4.3(b))。

最後に r_j が $i < k < j$ を満たす r_k と対合している状態を考えましょう。この場合、**ノット構造がないという仮定のもとでは**、図 4.3(c) に示すように、構造を $S_{i,k-1}$ と $S_{k,j}$ の 2 つの部分に分けて考えることができ、構造全体の自由エネルギーは、$S_{i,k-1}$ と $S_{k,j}$ がそれぞれ持つ自由エネルギーの和になります。ここでも RNA 中の 2 つの塩基が対合したときの自由エネルギーは他の場所の構造に関係なく一定という仮定を使うと、$S_{i,k-1}$ と $S_{k,j}$ の構造が持つ自由エネルギーの計算を独立に行うことが可能なので、構造全体の自由エネルギーを最小にするためには、$S_{i,k-1}$ と $S_{k,j}$ の両方の自由エネルギーを最小にすれば良いことになります。

しかし r_j がどの塩基と結合するのが適切か分かりませんので、i と j の間の全ての塩基について、r_j が結合した時の最小自由エネルギーを計算し、その中でも最も低かった自由エネルギーを選ぶことになります。これを式で表すと、

$$E(S_{i,j}) = \min\{E(S_{i,k-1}) + E(S_{k,j})\} \quad \text{for } i < k < j \tag{4.4}$$

となります。

さて実は、図 4.3(b) の r_j がどの塩基にも結合しないというケースは、(c) の分割で表すことができます。図 4.4 に示すように、構造を $S_{i,j-1}$ と、$S_{j,j}$ に分割するのです。この場合、$k = j$ です。ここで $E(S_{j,j}) = 0$ と定義して式 4.4 に代入すると、$k = j$ のときは $E(S_{i,j}) = E(S_{i,j-1}) + E(S_{j,j}) = E(S_{i,j-1})$ となり式 4.3 と同じになるので、結局式 4.3 と 4.4 を統合して、

$$E(S_{i,j}) = \min\{E(S_{i,k-1}) + E(S_{k,j})\} \quad \text{for } i < k \leq j \tag{4.5}$$

とすることができます。但し、$E(S_{j,j}) = 0$ です。式 4.4 と 4.5 の違いは、単に $k = j$ があるかないかだけです。

---

*4) 厳密には $\min_{S_{i,j}} E(S_{i,j}) = \alpha(r_i, r_j) + \min_{S_{i+1,j-1}} E(S_{i+1,j-1})$ となるが、本章では簡略化してこのように記述する。

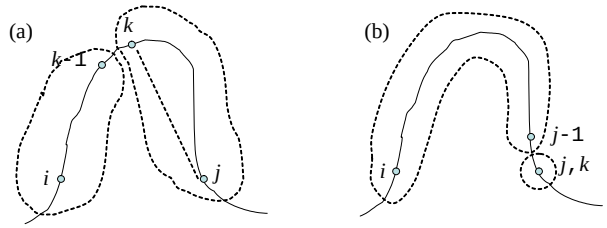


図 4.4　塩基 r_j が結合しない状態の分割による表現
(a) では構造 $S_{i,j}$ が $k-1, k$ の位置で分割されている。(b) において、$k = j$ にすることにより、r_j が結合していない状態を表現できる。

結局部分配列 $R_{i,j}$ が取りうる構造の自由エネルギーの中で最小のもの $E(S_{ij})$ は、式 4.2、4.5 で算出される最小自由エネルギーの中で最小のものになるのでこれら 2 つの式を統合して

$$E(S_{i,j}) = \begin{cases} 0 \quad \text{if} \qquad i = j \\ \min \begin{cases} E(S_{i+1,j-1}) + \alpha(r_i, r_j) \\ \min\{E(S_{i,k-1}) + E(S_{k,j})\} \text{ ただし } i < k \leq j \end{cases} \end{cases} \tag{4.6}$$

となります。与えられた配列全体の最小の自由エネルギーは $E(S_{1,n})$ となります。式 4.6 を関数 ES として実装すれば以下のようになるでしょう (演習問題 4.1)。

```
/* グローバル配列変数 r[] の中に塩基配列を入れておく */
/* alpha(char a, char b) は塩基 a,b が対合したときの
   自由エネルギーを返す関数 */

double ES(int i, int j){

  /* 変数の宣言などをここに書く */

/* 式 4.6 の最初の式 */
  if(i >= j)min = 0.0;
  else {
/* 式 4.6 の min の中の最初の式 */
    con = ES(i + 1, j - 1) + alpha(r[i], r[j]);
    disj_min = 99999;
/* 式 4.6 の min の中の 2 番目の式 */
    for(k = i + 1;k <= j;k ++){
         disj = ES(i, k - 1) + ES(k, j);
         if(disj_min > disj){
                disj_min = disj;
                disj_min_k = k;
         }
    }
    if(con < disj_min)min = con;
    else min = disj_min;
  }
  return min;
}
```

	1	2	3	4	5	6	7	8	9	10
1		0	0	−2	−2	−4	−6	−9	−9	−9
2			0	−2	−2	−4	−6	−6	−6	−6
3				−2	−2	−4	−4	−4	−4	−4
4					−2	−2	−2	−2	−2	−2
5						−2	−2	−2	−2	−2
6							0	0	0	0
7								0	0	0
8									0	0
9										0
10										

表 4.1　$E(S_{i,j})$ を記録する行列 E

配列”gttataacac”を対象とし、A-U の対合は-2、C-G の対合は-3 とした。また、RNA はいくらでもきつく曲がることができるとした。

4.5　計算の重複の回避

$1 \leq i < j \leq n$ に対して、$E(S_{i,j})$ は決まった値を持つはずです。従って一度ある $E(S_{i,j})$ を計算してしまえば、同じ $E(S_{i,j})$ を 2 度以上計算する必要はありません。しかし式 4.6 をそのまま使うと、同じ部分に対して複数回計算が行われてしまいます。実際部分構造 $S_{s,t}$ が計算されるのは、部分構造 $S_{s,u}$ が $S_{s,t}, S_{t+1,u}$ に分割されたときや、$S_{s,v}$ が、$S_{s,t}, S_{t+1,v}$ に分割されるときなど沢山のケースがあります。そこで、一度計算した $E(S_{i,j})$ の値を例えば $n \times n$ の行列 E に記録し[*5)]、二度目から E に保存した値を使って再帰が起こらないようにすると、計算速度を飛躍的に向上させることができます。

また別法として、$E(S_{i,j})$ を求める際にどの部分構造の最小自由エネルギーがあらかじめ分かっていなくてはならないかを考慮し、再帰を使わずに効率よく $E(S_{i,j})$ を求めることも可能です。

基本的には A.3 で学習した方法と一緒ですので、復習の上、演習問題 4.2, 4.3 に挑戦してみて下さい。

表 4.1 に”gttataacac”を二次構造予測にかけ、行列 E に値を記録していった結果を示します。A-U の対合は-2、C-G の対合は-3 とし、RNA はいくらでもきつく曲がることができるとしました。

---

*5)　実際に値を入れるのは行列の右上半分だけ、つまり $i < j$ のところだけで済む。

	1	2	3	4	5	6	7	8	9	10
1		2	2	2	2	2	2	XX	9	9
2			3	3	3	3	XX	8	8	8
3				XX	4	5	5	5	5	5
4					XX	5	5	5	5	5
5						XX	7	7	7	7
6							7	7	7	7
7								8	8	8
8									9	9
9										10
10										

表 4.2　構造 $S_{i,j}$ の最小自由エネルギーが求まったときに行った操作 (最適な位置での構造の分割または両端塩基の結合 (XX)) を記録する行列 D

4.6　二次構造の決定

これまで与えられた RNA の一次配列に対して最小の自由エネルギーを求める方法を考えてきました。ここからは、最小の自由エネルギーに対応する構造を求める方法について考えてゆきましょう (演習問題 4.4)。

最小自由エネルギーを求める過程で、塩基同士をどのように結合させていったか、また RNA の構造をどの位置で分割していったかをまず記録してゆきます。より具体的には $n \times n$ の行列 D を用意し、構造 $S_{i,j}$ の最小自由エネルギー $E(S_{i,j})$ は

1. r_i, r_j を結合させたときに得られたか
2. 構造 $S_{i,j}$ を k の位置で分割したときに得られたか

を $D_{i,j}$(D の i 行目、j 列目) に記録してゆきます。例えば、部分構造 $S_{i,j}$ の最小自由エネルギーが両端の塩基を結合させたときに得られた場合、$D_{i,j}$ に”XX”を記録します。また最小自由エネルギーが構造を $S_{i,k-1}$ と $S_{k,j}$ に分割したときに得られた場合、$D_{i,j}$ に k を記録します[*6)]。

表 4.2 に先ほどの配列の最小自由エネルギーを求める過程で記録に使用した行列 D の中味を示します。行列の 1 行目 10 列目を見ると、全体の構造 $S_{1,10}$ の自由エネルギーが最小になるのは、構造を $S_{1,8}$ と $S_{9,10}$ に分割したときであることが分かります。

次に行列 D から二次構造を表す行列 C を計算します。行列 C が表現する意

*6)　分割の場合、$i < D_{i,j} \leq j$ となるので、結合のときは $D_{i,j}$ に i を入れても分割と結合を区別できる。コンピュータは数値データの方が扱いやすいため、この記録方法が効果的 (演習問題 4.4)。

	1	2	3	4	5	6	7	8	9	10
1		0	0	0	0	0	0	1	0	0
2			0	0	0	0	1	0	0	0
3				1	0	0	0	0	0	0
4					0	0	0	0	0	0
5						1	0	0	0	0
6							0	0	0	0
7								0	0	0
8									0	0
9										0
10										

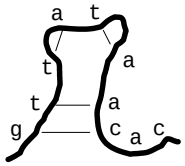


表 4.3　二次構造を表す行列 C

表の左下にこの行列が表している RNA の二次構造を示した。

味は、図 4.1(d) と同じです。まず、RNA 全体の最小自由エネルギー $E(S_{1,n})$ が求まったときに、どのような操作をしたかを $D_{1,n}$ より得ます。$D_{1,n}$ =”XX”、つまり r_1 と r_n を結合させていれば、$C_{1,n} = 1$ として、次にその 1 つ内側の構造 $S_{2,n-1}$ を調べます。$D_{1,n}$ に数値 k が入っていれば、構造を $k-1, k$ の位置で分割したということなので、次に構造 $S_{1,k-1}$ と $S_{k,n}$ を調べてゆきます。

例えば表 4.2 より全体の構造 C を計算するためには、まず $D_{1,10}$ を見ます。値が 9 なので構造を $S_{1,8}$ と $S_{9,10}$ に分割し、次に $D_{1,8}$ と $D_{9,10}$ を調べます。後者の方は次に $S_{9,9}$ と $S_{10,10}$ に分割されてしまい、1 塩基ずつになってしまうのでそこで処理が終わりです。前者の方は $D_{1,8}$ が”XX”になっているので、$C_{1,8} = 1$ とすることによって r_1 と r_8 の塩基を結合させます。そしてその内側の構造 $S_{2,7}$ の $D_{2,7}$ を調べます。

D から C を求める正確なアルゴリズムは以下の通りです。これをプログラムとして実装してみましょう (演習問題 4.4)。

DtoC(i, j):行列 D を参照し、RNA 配列の一部 (i, j) の二次構造 C を求める

1. $i \geq j$ なら処理なし
2. そうでなければ、$D_{i,j}$ =XX なら、$C_{i,j}$ を 1 にして、DtoC$(i+1, j-1)$ を呼ぶ（塩基の結合）
3. そうでなければ、DtoC$(i, D_{i,j}-1)$ を呼んだ後、DtoC$(D_{i,j}, j)$ を呼ぶ (構造の分割)

表 4.2 の D をもとに求めた C を表 4.3 に示します。

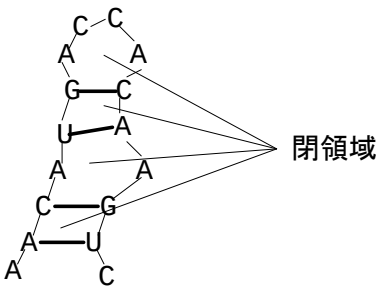


図 4.5　RNA 二次構造の閉領域

4.7　Zuker 法

これまでは各塩基対に対して自由エネルギーを決め、構造内の塩基対の自由エネルギーの総和が構造全体の自由エネルギーであるという非常に単純な自由エネルギー計算を行っていました。しかしこの計算法は RNA 構造の自由エネルギーの近似計算を行う上で必ずしも十分とは言えません。そこで多くの二次構造予測ではより複雑な近似を行っています。この節では Zuker 法 [15] で使われている近似計算をまず紹介しましょう。

Zuker 法では閉領域 (face) と呼ばれる単位で自由エネルギーの計算が行われます。閉領域とは図 4.5 で考えると、塩基配列をつなぐ線（共有結合を表す）と、塩基間の対合を表す線（水素結合を表す）で囲まれた領域のことです。厳密に言えば、閉領域は以下のように定義されます。

1. $i,u,v,j(i<u<v<j)$ が閉領域であるとは、$(i,j)\in S$ かつ $(u,v)\in S$ であり、$i<s<u,v<t<j$ を満たす全ての s,t に対して、$(s,t)\notin S$ が成立するときである[*7)]。このとき閉領域の大きさは、$(u-i-1,j-v-1)$ である。
2. (i,j) が閉領域であるとは、$i<s<t<j$ を満たす全ての全ての s,t に対して、$(s,t)\notin S$ が成立するときである。このとき閉領域の大きさは、$j-i-1$ である。

閉領域は図 4.6 に示すように、その形状によっていくつかに分類されます。1 で定義される閉領域のうち、大きさが (0,0) のものをステム構造、大きさが $(0,p)$ または $(p,0)$ で、$p>0$ のものをバルジ構造、大きさが (p,q) で、$p>0,q>0$ のものを内部ループと呼びます。また 2 で定義される閉領域をヘアピン構造と呼びます。

---

*7)　$i<s<u,v<t<j$ を満たす s,t がないときは、$(i,j)\in S$ かつ $(u,v)\in S$ が成立すれば、i,u,v,j が閉領域であると見なされる。この場合この閉領域はステム構造またはバルジ構造となる。

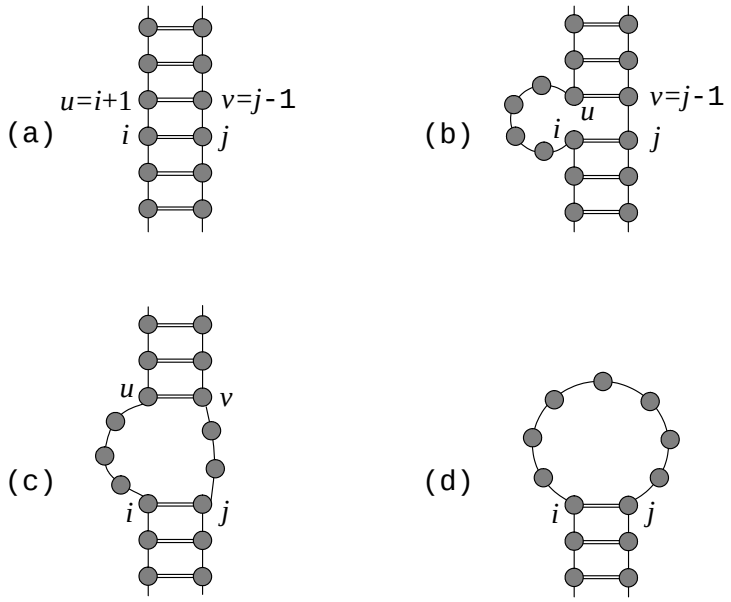


図 4.6　RNA の二次構造の部分構造の分類
灰色の印は塩基を表し、横の二重線は塩基対を表す。(a) ステム構造、(b) バルジ構造、(c) 内部ループ、(d) ヘアピン構造。

ステム構造

	3'GU	3'AU	3'UA	3'CG	3'GC
5'GU	−0.5	−0.5	−0.7	−1.9	−1.9
5'AU	−0.5	−0.9	−0.9	−2.1	−1.7
5'UA	−0.7	−1.1	−0.9	−2.3	−1.8
5'CG	−1.5	−1.8	−1.7	−2.9	−3.4
5'GC	−1.3	−2.3	−2.1	−3.4	−2.9

ヘアピン構造

	0	1	2	3	4	5	6	7	8	9
0				7.4	5.9	4.4	4.3	4.1	4.1	4.2
10	4.3		4.9		5.6		6.1		6.7	
20	7.1					8.1				
30	8.9									

バルジ構造

	0	1	2	3	4	5	6	7	8	9
0		3.3	5.2	6	6.7	7.4	8.2	9.1	10	10.5
10	11		11.8		12.5		13		13.6	
20	14					15				
30	15.8									

内部ループ

	0	1	2	3	4	5	6	7	8	9
0	0.0	0.8	0.8	1.3	1.7	2.1	2.5	2.6	2.8	3.1
10	3.6		4.4		5.1		5.6		6.2	
20	6.6					7.6				
30	8.4									

表 4.4　RNA の部分構造に対する自由エネルギー [16](kcal/mol)

”ステム構造”の表では、一番左の列に 5' 側の塩基対を、一番上の行に 3' 側の塩基対を示し、対応する自由エネルギーを記入した。他の表の一番左の列 (10 の位) と一番上の列 (1 の位) に閉領域の大きさを示し、対応する自由エネルギーを記入した。

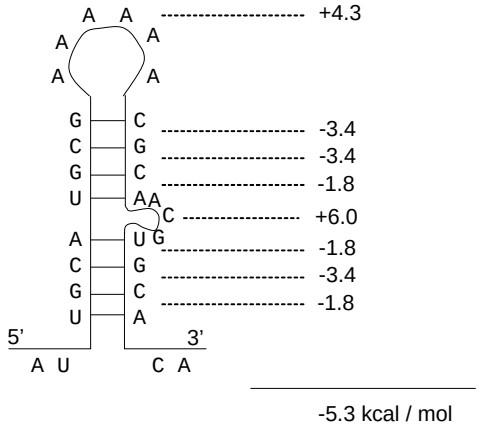


図 4.7　Zuker 法における自由エネルギーの計算

ステム構造は RNA の構造を安定化するので、負の自由エネルギーが与えられます。逆に、バルジ構造、内部ループ、ヘアピン構造は RNA の構造を不安定にするものと考えられ、正の自由エネルギーが与えられます。閉領域がその形状に応じてどれくらいの自由エネルギーを持つかは生化学的な実験で調べられています (表 4.4)。

Zuker 法では RNA 全体の自由エネルギーは、全ての閉領域の自由エネルギーの和であると定義します。例えば図 4.7 のような二次構造に対しては、各部分の自由エネルギーを表 4.4 で計算し、それを全て合計して -5.3 kcal/mol となります。そして最適な構造 $S_{1,n}$ は、このように定義された RNA 全体の自由エネルギーが最小の値 $E(S_{1,n})$ になるものです。これをどのように求めればよいかを考えてゆきましょう。

まず $S_{1,n}$ の部分構造 $S_{i,j}$ の最小自由エネルギー $E(S_{i,j})$ の求め方から考えましょう。5' 端側の塩基 r_i が対合していないと仮定します。すると、$E(S_{i,j}) = E(S_{i+1,j})$ となります。次に 3' 端側の塩基 r_j が対合していないと仮定します。すると同様に $E(S_{i,j}) = E(S_{i,j-1})$ となります。そして r_i と r_j が互いにでなく、別の塩基と対合していると仮定します。すると $E(S_{i,j}) = \min\{E(S_{i,k-1}) + E(S_{k,j})\}$(ただし、$i+1 < k < j$) となります。最後に (r_i, r_j) が対合する場合が考えられます。(r_i, r_j) が対合するような構造を $L_{i,j}$ とすれば、結局

$$E(S_{i,j}) = \min \begin{cases} E(S_{i+1,j}) \\ E(S_{i,j-1}) \\ E(L_{i,j}) \\ \min\{E(S_{i,k-1}) + E(S_{k,j})\} \quad \text{for} \quad i+1 < k < j \end{cases} \tag{4.7}$$

という再帰式を立てることができます。さて $E(L_{i,j})$ について詳しく考えましょう。(r_i, r_j) が対合している場合、それは図 4.6 に示すような構造の一部であるか、図 4.10 に示すような分岐構造の一部になっているはずです。図 4.6 に示すような構造であれば、これは先に定義した閉領域に該当し、その部分の自由エネルギーが一意に決まります。そこで図 4.8 に示すように、ヘアピンの自由エネルギーを $\xi(j-i-1)$ とし、それ以外のヘアピン、ステム構造、内部ループ、バルジ構造の自由エネルギーを $e(i,j,u,v,R)$ で表しましょう。ステム構造のときは、塩基配列 R の中の塩基 r_i, r_j, r_u, r_v が自由エネルギーに関係しますが、内部ループやバルジ構造の場合は閉領域の大きさである $(u-i-1, j-v-1)$ のみが自由エネルギーに関係します。

$L_{i,j}$ がそのままヘアピン構造となる場合は、$E(L_{i,j}) = \xi(j-i-1)$ となります。ステム構造か、内部ループ、バルジ構造になる場合は、図 4.9 に示すように、$E(L_{i,j}) = e(i,j,u,v,R) + E(L_{k_1,k_2})$ となります。そして分岐構造の一部であれば、図 4.10 に示すように、$E(L_{i,j}) = E(S_{i+1,k-1}) + E(S_{k,j-1})$(但し、$i+2 < k < j-1$) となります。ここで分岐構造自体の自由エネルギーは 0 とします。以上の関係をまとめると、

$$E(L_{i,j}) = \min \begin{cases} \xi(j-i-1) \\ \min\{e(i,j,k_1,k_2,R) + E(L_{k_1,k_2})\} \\ \qquad\qquad \text{for} \quad i < k_1 < k_2 < j \\ \min\{E(S_{i+1,k-1}) + E(S_{k,j-1})\} \\ \qquad\qquad \text{for} \quad i+2 < k < j-1 \end{cases} \tag{4.8}$$

となります。但し $e(i,j,k_1,k_2,R)$ は r_i と r_i、r_{k_1} と r_{k_2} がそれぞれ結合したときの自由エネルギーを表します。バルジループ、内部ループ、ヘリカル領域の自由エネルギーが全て関数 e で表現されています。

演習問題

4.1 $E(S_{i,j})$ の値を求める関数をプログラムとして実装しましょう。

4.2 演習問題 4.1 のプログラムを改良し、$E(S_{i,j})$ の値を $n \times n$ 行列 `e_matrix[][]` に逐次入れて、随時参照することにより、高速化しましょう。

4.3 再帰を使わずに $E(S_{i,j})$ を求める関数を実装しましょう。

4.4 最小の自由エネルギーに対応する二次構造を求めるプログラムを作成しましょう。本文中の行列 D に該当する二次元配列 `d_matrix[][]` と C に該当する二次元配

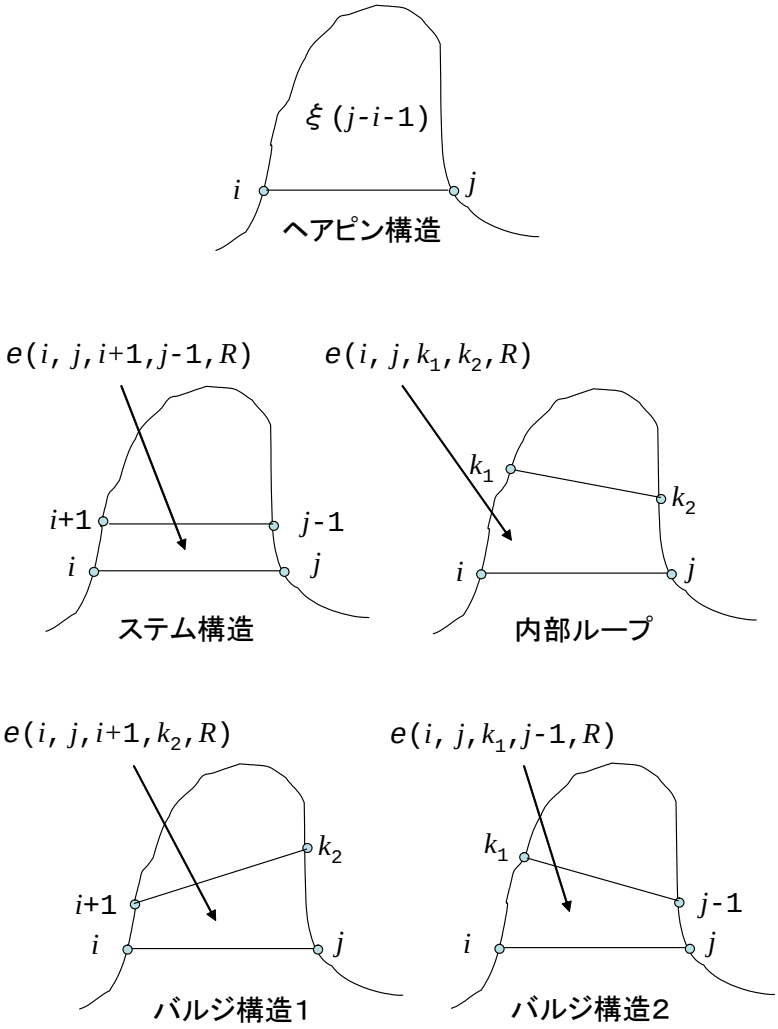


図 4.8　閉領域の自由エネルギーとそれを求める関数 e

各閉領域のタイプと、その閉領域の自由エネルギーを関数 ξ および e を使って示した。

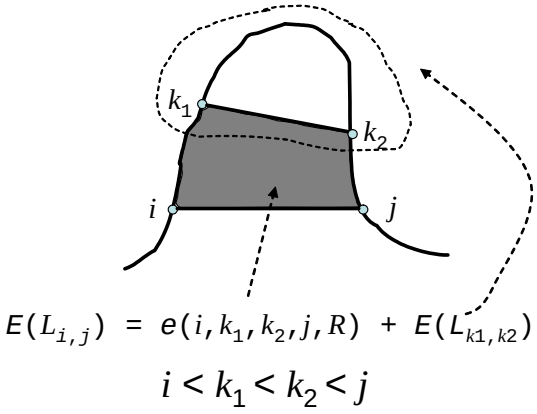


図 4.9　構造 $L_{i,j}$ の末端がステム構造または内部ループ、バルジ構造になるときの最小自由エネルギーを求める再帰式

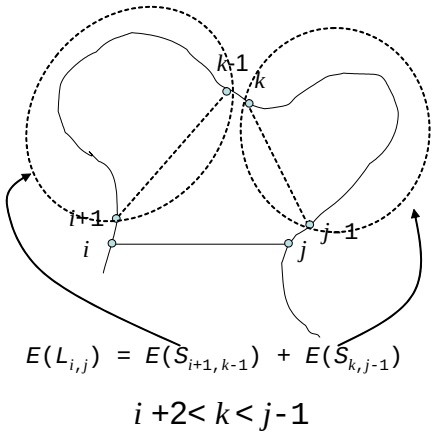


図 4.10　構造 $L_{i,j}$ の末端が分岐構造の一部になるときの最小自由エネルギーを求める再帰式

列 `c_matrix[][]` を用意し、節 4.6 で解説した関数 DtoC を参考に、`d_matrix[][]` から `c_matrix[][]` を再帰的に求める関数 `d_to_c` を作ると効率良く構造を求めることができるでしょう。なお、`d_matrix[i][j]` が k(i<k≤j) のときは、構造が k-1 と k のところで分割されたことを示し、`d_matrix[i][j]` が i のときは、i と j が結合したことを示すようにするとよいでしょう。

4.5 節 4.4 で取り扱った簡単な RNA 二次構造の予測方法では、分割を表現する式 4.5 で両端の塩基が結合していない状態を取り扱うことができました。Zuker 法では、$E(S_{i,j}) = E(S_{i+1,j}), E(S_{i,j}) = E(S_{i,j-1})$ のように、両端の塩基が結合していない状態を明確に表現する必要があります。何故でしょうか？

第 5 章
遺伝子発現データのクラスタリング

クラスタリングは類似したデータを集めることです。近年酵母菌を初めとした様々な生物種で多量の遺伝子発現データが公開されていますが、それらの中から遺伝子の発現制御関係を抽出するなど何らかのデータマイニングをする上で、クラスタリングは最も基本的な手法と言えます。そこで本章では、遺伝子発現データのクラスタリングの基本について学習します。

5.1 遺伝子発現データ

１つの生物が持つ遺伝子の数は大腸菌で約 4,000、酵母菌で約 6,000、ヒトでは数万と言われています。その中には常に発現しているような遺伝子もあれば、時期特異的に発現する遺伝子もあり、また多細胞生物では、組織特異的に発現するものもあります。多くの遺伝子は、細胞にとって必要なときに発現すると考えられます。

例えば細胞分裂を起こすときには、細胞分裂に関係した遺伝子が発現し、大腸菌に熱を与えたとき、ヒートショック関連の遺伝子が発現して、細胞を守ります。

逆に考えると例えば、細胞に紫外線を当てたときに発現する遺伝子群の多くは細胞を紫外線から守る機能を持っているのではないかと考えられます。そこである生物が持つ全ての遺伝子の発現量をモニタリングすることができれば、紫外線を当てたときに発現量が上がる遺伝子群を全て抽出することにより、紫外線から細胞を守る働きがあると思われる遺伝子の候補を抽出できることになります。

マイクロアレイ (図 5.1) という装置が開発されて以来、多くの遺伝子の発現量を同時に計測できるようになり、酵母菌を初め、多数の生物についてゲノム全体の遺伝子の発現量の測定が行われ、遺伝子発現データが公開されています [17], [18]。

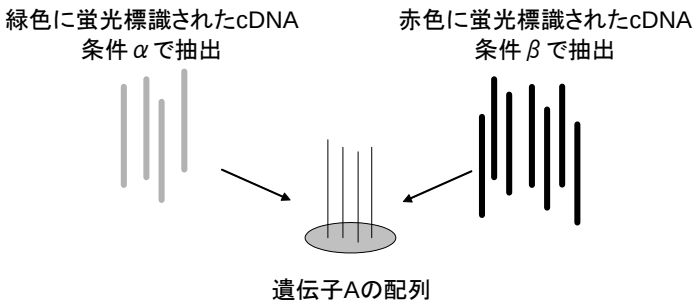


図 5.1　遺伝子の発現量の比を測るマイクロアレイ

遺伝子 A が条件 α と比べて条件 β でどれくらい発現しているかをマイクロアレイを使って調べる方法を示す。チップ上に遺伝子 A のプローブとなる配列（相補的な配列）を貼り付けておく。条件 α で得られた転写産物を cDNA に逆転写し、緑色に蛍光標識を付ける。また条件 β で得られた転写産物も cDNA に逆転写し、こちらは赤色に蛍光標識を付ける。これらの標識した cDNA をチップ上でハイブリさせると、条件 α における遺伝子 A の発現量の方が多ければチップは緑色の蛍光を発し、条件 β における遺伝子 A の発現量の方が多ければ赤色の傾向を発する。こうして条件 α と β における遺伝子 A の発現量の比を知ることができる。

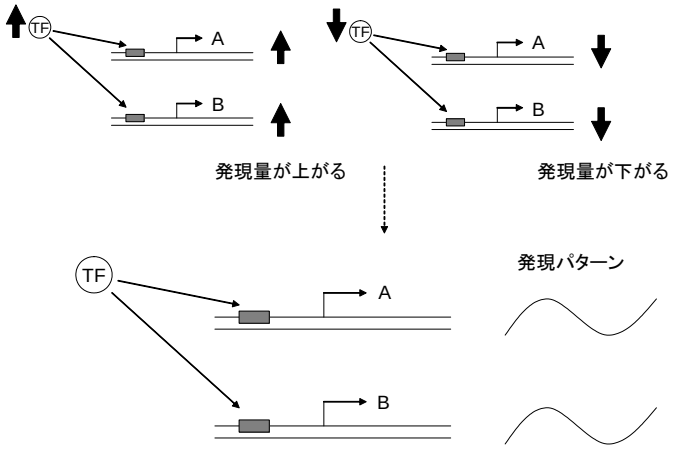


図 5.2　転写因子と発現パターンの関係

5.2　発現相関の定量化

多数の遺伝子について多量の遺伝子発現データが与えられたとき、遺伝子の発現制御の構造を推定することが可能になります。例えば、図 5.2 のように、ある転写因子 (TF) が 2 つの遺伝子の発現を制御しているとき、その転写因子が発現すれば、2 つの遺伝子の発現も同時に上がり、転写因子が発現しなくなれば、2 つの遺伝子の発現も同時に下がることになります[*1)]。

このように転写制御において、緊密な関係のある遺伝子群の発現パターンは類似する傾向にあります。逆に類似した発現パターンを示す遺伝子群は転写制御において密接な関係にあることが推測されます。そこで多量な遺伝子の発現データから類似した発現パターンを示すものを抽出することができれば、近い制御関係にある遺伝子をまとめることが可能です。

では、遺伝子の発現パターンの類似性を測定するためにはどのような尺度が考えられるでしょうか?まず考えられるのは、ユークリッド距離です。今、遺伝子 1 の発現データ $x_{11}, x_{12}, x_{13}, \cdots, x_{1n}$ と、遺伝子 2 の発現データ $x_{21}, x_{22}, x_{23}, \cdots, x_{2n}$ が与えられたとすると、ユークリッド距離 d は

$$d = \sqrt{\sum_{k=1}^{n} (x_{1k} - x_{2k})^2}$$

と定義されます。これは図 5.3 に示すように、2 つの遺伝子の発現量の多次元

*1)　リプレッサーの場合は逆に、発現するとそのターゲットとなる遺伝子の発現量は下がる。

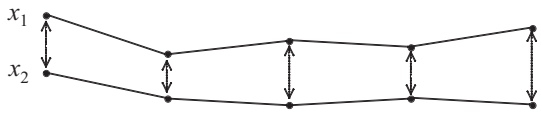


図 5.3　ユークリッド距離

2 つの遺伝子 x_1 および x_2 の発現パターンの例を示した。縦軸が各遺伝子の発現レベルを表し、横軸はその発現レベルが観測された条件 (時間や組織、栄養状態など) を表す。同じ条件における遺伝子の発現レベルの距離を矢印付きの線で表した。ユークリッド距離は、この距離の二乗の総和の平方根である。

上の「距離」を表し、小さければ小さいほど発現量が似ていることを表します。ユークリッド距離は発現レベルの絶対値が意味を持つような場合に有効です。

しかしユークリッド距離を使うと、波形の形が似ていても発現レベルが大きく異なる場合、距離が大きくなってしまいます。そこでもう 1 つよく使われるのがピアソンの相関係数で、これを使うと、遺伝子発現パターンの波形の類似性を定量することが可能です。2 つの遺伝子の発現の相関は以下のように求めることができます。

遺伝子 1 の発現データ $x_{11}, x_{12}, x_{13}, \cdots, x_{1n}$ と、遺伝子 2 の発現データ $x_{21}, x_{22}, x_{23}, \cdots, x_{2n}$ が与えられたとします。まず、平均 μ と分散 σ^2 を以下の式により求めます*2)

$$\mu_i = \frac{1}{n}\sum_{k=1}^{n} x_{ik} \qquad \sigma_i^2 = \frac{\sum_{k=1}^{n}(x_{ik}-\mu_i)^2}{n}$$

ただし i は遺伝子番号 (この例では 1 または 2) を表します。次に x_1 と x_2 のデータの標準化を行います。

$$z_{ij} = \frac{x_{ij}-\mu_i}{\sigma_i}$$

こうすると、元の遺伝子発現データがどのような分布をしていても、それを変換した z_i は平均が 0、分散が 1 になります。そして相関係数 r を以下の式で計算します。

$$r = \frac{1}{n}\sum_{k=1}^{n} z_{1k} z_{2k}$$

z_{1k} と z_{2k} の発現パターンが類似していれば、z_{1k} と z_{2k} が同時に正または負になる機会が多くなり、その積の和である r は正に大きくなることが分かりま

*2)　遺伝子発現データのクラスタリングの先駆的な研究を行った Eisen ら [19] はこの式で平均を 0 として発現相関を計算している。なお、遺伝子の発現データを標本として捉える場合、標本平均は $\overline{x_i} = \frac{1}{n}\sum_{k=1}^{n} x_{ik}$、標本分散は $s^2 = \frac{1}{n-1}\sqrt{\sum_{k=1}^{n}(x_{ik}-\overline{x_i})}$ で求める。また標本相関係数は、$\frac{1}{n-1}\sum_{k=1}^{n}\frac{x_{1k}-\overline{x_1}}{s_1}\frac{x_{2k}-\overline{x_2}}{s_2}$ となる。

す。逆に z_{1k} と z_{2k} の発現パターンがちょうど正反対であれば、z_{1k} と z_{2k} の符号が逆になる機会が多くなり、その積の和である r は負に大きくなります。また z_{1k} と z_{2k} の間に相関がなければ、$z_{1k}z_{2k}$ は正の値も負の値もとるようになり、その和は 0 に近くなります。

z_1 も z_2 も平均が 0、分散が 1 なので、$t \in R$ について、

$$\frac{1}{n}\sum_{k=1}^{n}(z_{1i} - tz_{2i})^2 = (t-r)^2 - r^2 + 1 \geq 0$$

が成立します。任意の $t \in R$ に対して $(t-r)^2 \geq 0$ なので、$-1 \leq r \leq 1$ となります。

図 5.4 に 2 つの遺伝子の発現パターンの例と、その相関係数を載せました。この図より、

- 発現パターンが似ているとき、相関係数 r は 1 に近づく
- 発現パターンに相関がないとき、相関係数 r は 0 に近づく
- 発現パターンが逆になるとき、相関係数 r は-1 に近づく

ということが分かると思います。

5.3 遺伝子発現データの階層的クラスタリング

前節で遺伝子間の類似性の定量に相関係数などの指標が使えることを説明しました。それでは、類似性の高い遺伝子群が同じグループに含まれるようにグルーピングをするにはどうすればいいでしょうか？

このように似た性質のデータをグループごとに集めることをクラスタリングと呼びます。効果的なクラスタリングを行う 1 つの方法は階層的クラスタリングです [19], [20]。このクラスタリングでは最も似ている（発現相関が高い) データの対から順番にクラスタリングを行い、階層構造を構築します。具体的な方法を 4 つの遺伝子発現量の架空データを用いて図 5.5 を使いながら説明します。

1. 全通りの遺伝子間の発現相関を求める（その結果を図の Step 1 の表に示す)。この例では相関係数を用いる。
2. 最も発現相関の高かった遺伝子の組を 1 つに統合 (Step 1 の右)。図の例では Data 2 と Data 3 の発現相関が最も高いので、これを 1 つにまとめる (図の右の木構造参照)。
3. 再び全通りの組の発現相関を求める (Step 2 の左の表)。
4. 再び遺伝子またはクラスターの組を統合。この例では、Data0 と Data1 が統合される (Step 2 の右)。
5. Data0 + Data1 のクラスターと、Data2 + Data3 のクラスターの発現相関は-0.44(Step 3 の左)。最後に残ったこの 2 つのクラスターが統合される (Step 3 の右)。

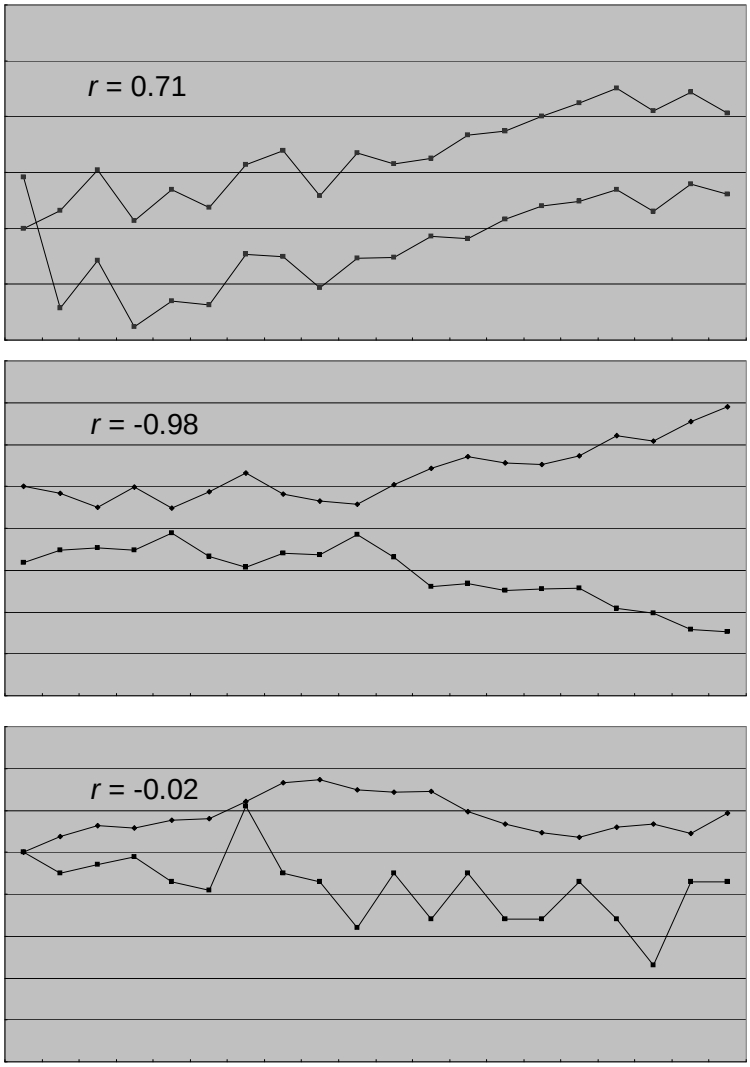
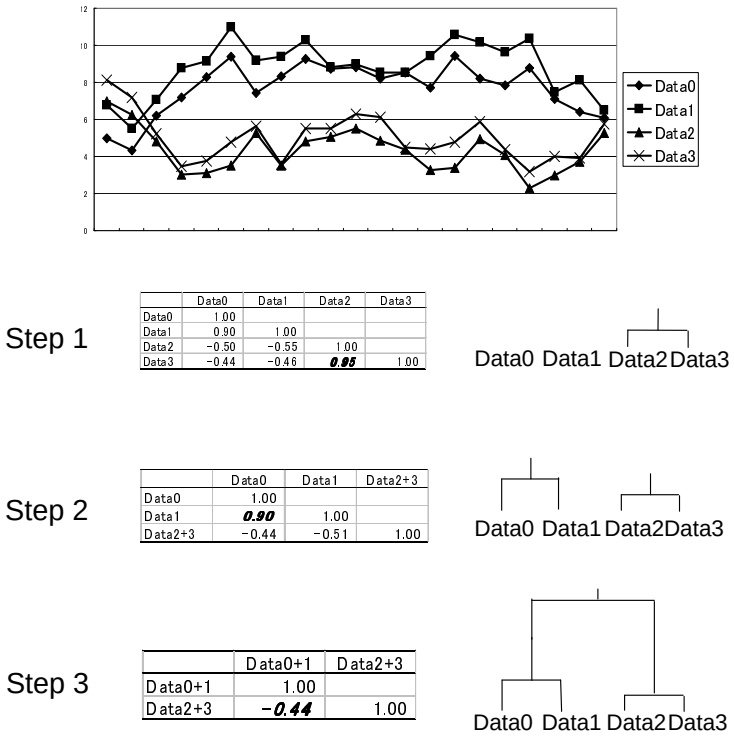


図 5.4　曲線の類似性と相関係数



	Data0	Data1	Data2	Data3
Data0	1.00			
Data1	0.90	1.00		
Data2	−0.50	−0.55	1.00	
Data3	−0.44	−0.46	***0.95***	1.00

	Data0	Data1	Data2+3
Data0	1.00		
Data1	***0.90***	1.00	
Data2+3	−0.44	−0.51	1.00

	Data0+1	Data2+3
Data0+1	1.00	
Data2+3	***−0.44***	1.00

図 5.5　階層的クラスタリングの例

一番上に 4 つのデータ (Data0 〜 Data3) を示す。左側の表が発現相関、右の図が階層的クラスタリングの途中経過を表す。

統合を繰り返していくと、最終的に全てが 1 つの枝に統合されるので (Step 3)、そこでアルゴリズムは終了となります。

遺伝子間の発現相関は前述の相関係数を使って計算することができます。ではクラスターと遺伝子、あるいはクラスター間の遺伝子の発現相関はどのように求めればよいでしょうか？最も単純な方法は、最短距離法です。これは例えばクラスター A と B の発現相関を求める場合、A に含まれる遺伝子と B に含まれる遺伝子の全通りの相関係数を計算し、その中で最も高い相関係数がクラスター A,B 間の相関係数になります[*3)]。Data0 + Data1 のクラスターと Data2 + Data3 のクラスターの発現相関を計算する場合、全通りの相関係数は Data0 と Data2 の −0.50、Data0 と Data3 の −0.44、Data1 と Data2 の −0.55、Data1 と Data3 の −0.46 の 4 つになり、最も高いのは −0.44 なので、これがこの 2 つのクラスターの発現相関となります。

最短距離法を使った階層的クラスタリングの実装例を以下に示します。この例では CLUSTER という構造体の配列 cl で各クラスターに含まれる遺伝子を管理しています。

```
#define N_DATA 4 /* データ数 */
#define DIM 21

/* クラスター情報 */
struct CLUSTER {
  int n; /* データ数 */
  int gene[N_DATA]; /* 保持しているデータ番号の集合 */
};

/* プロトタイプ宣言 */
double  corr_single(double a[], double b[], int n);
double  corr_clust(struct CLUSTER& a, struct CLUSTER& b,
                   double d[N_DATA][N_DATA]);
void merge(struct CLUSTER& a, struct CLUSTER& b, struct CLUSTER& merged);

/* n 次元データ a, b の相関係数を計算する */
double corr_single(double a[], double b[], int n){
  int i;
  double d = 0.0;
```

*3) 他に全通りの相関係数の中で最も低い相関係数をクラスター間の相関係数とする最長距離法や、全通りの相関係数の平均をクラスター間の相関係数とする平均距離法などがある。詳しくは文献 [21] などを参照。

```
  double mean_a, mean_b;
  double var_a, var_b;
  double corr;

  for(mean_a = 0, i = 0;i < n;i ++)mean_a += a[i] / n;
  for(mean_b = 0, i = 0;i < n;i ++)mean_b += b[i] / n;
  for(var_a = 0, i = 0;i < n;i ++)
    var_a += (a[i] - mean_a)*(a[i] - mean_a) / n;
  for(var_b = 0, i = 0;i < n;i ++)
    var_b += (b[i] - mean_b)*(b[i] - mean_b) / n;
  for(corr = 0, i = 0;i < n;i ++)
    corr += (a[i] - mean_a) / sqrt(var_a)
      * (b[i] - mean_b) / sqrt(var_b) / n;

  return corr;
}

/* クラスター a, b の要素間の相関を計算する */
double corr_clust(struct CLUSTER& a, struct CLUSTER& b,
                  double d[N_DATA][N_DATA]){
  int i, k;
  double max;

  max = d[ a.gene[0] ][ b.gene[0] ];
  for (i = 0; i < a.n; i ++){
      for (k = 0; k < b.n; k ++){
          if (max < d[ a.gene[i] ][ b.gene[k] ])
              max = d[ a.gene[i] ][ b.gene[k] ];
        }
    }
  return max;
}

/* クラスター a, b を統合し、結果をクラスター new に入れる */
void merge(struct CLUSTER& a, struct CLUSTER& b, struct CLUSTER& merged){
  int i, n;

  n = 0;
```

```
  for (i = 0; i < a.n; i ++)
      merged.gene[ n++ ] = a.gene[i];
  for (i=0; i < b.n; i++)
      merged.gene[ n++ ] = b.gene[i];
  merged.n = n;
}

/* 次元数 dim のデータ群 data を階層的クラスタリング
   データ数 N_DATA はグローバル変数として与えられる */
void h_cluster(double data[N_DATA][DIM]){
  int i, j, n;
  double d[N_DATA][N_DATA];   /* データ間の相関行列 */
  double corr, corr_max;      /* データ間の相関 */
  int c1, c2;

  static struct CLUSTER cl[N_DATA]; /* クラスター情報 */
  static struct CLUSTER w;     /* ワーク用 */

  /* クラスター情報の初期化 */
  for (i=0; i < N_DATA; i++){
      cl[i].n = 1;
      cl[i].gene[0] = i;
    }

  /* 相関行列の作成 */
  for (i = 0; i < N_DATA; i++)
      for (j = 0; j < N_DATA; j++)
        d[i][j] = corr_single(data[i], data[j], DIM);

  /* 階層的クラスタリング */
  n = N_DATA - 1;
  while (n > 0){
    corr_max = -1.0;
    for (i = 0; i < n; i ++){
        for (j = i + 1; j <= n; j ++){
              corr = corr_clust(cl[i], cl[j], d);
            if (corr > corr_max){
                corr_max = corr;
```

```
                c1 = i, c2 = j;
              }
            }
          }

    printf("[ Merge %d ] Merging the following two clusters:\n",
           N_DATA - n - 1);
    printf("(1) Cluster containing data #");
    for(i = 0;i < cl[ c1 ].n;i ++){
      printf("%d", cl[ c1 ].gene[i]);
      if(i < cl[ c1 ].n - 1)putchar(',');
    }
    printf("\n(2) Cluster containing data #");
    for(i = 0;i < cl[ c2 ].n;i ++){
      printf("%d", cl[ c2 ].gene[i]);
      if(i < cl[ c2 ].n - 1)putchar(',');
    }
    printf("\nDistance between these two clusters = %lf\n\n", corr_max);

    merge(cl[ c1 ], cl[ c2 ], w); /* クラスターを統合 */
    cl[ c1 ] = w; /* クラスター c2 を c1 へ統合して格納 */
    cl[ c2 ] = cl[n]; /* クラスター c2 を破棄し、一番後ろのクラスター
を代わりに格納 */
    n--; /* クラスターを 1 つ減らす */
  }
}
```

図 5.5 のデータを用いたプログラムの実行結果を以下に示します。

```
[ Merge 0 ] Merging the following two clusters:
(1) Cluster containing data #2
(2) Cluster containing data #3
Distance between these two clusters = 0.947762
```

この時点でデータ 2 とデータ 3 の相関係数が 0.95 と最も高かったので、1 つのクラスターとして統合されました。

```
[ Merge 1 ] Merging the following two clusters:
```

```
(1) Cluster containing data #0
(2) Cluster containing data #1
Distance between these two clusters = 0.897216
```

次にデータ 0 とデータ 1 の相関係数が 0.90 と最も高かったので、1 つのクラスターとして統合されました。

```
[ Merge 2 ] Merging the following two clusters:
(1) Cluster containing data #0,1
(2) Cluster containing data #2,3
Distance between these two clusters = -0.438644
```

最後にデータ 0,1 を含むクラスターと、データ 2,3 を含むクラスターが 1 つのクラスターとして統合されました。このときのクラスター間の相関係数は-0.44 です。

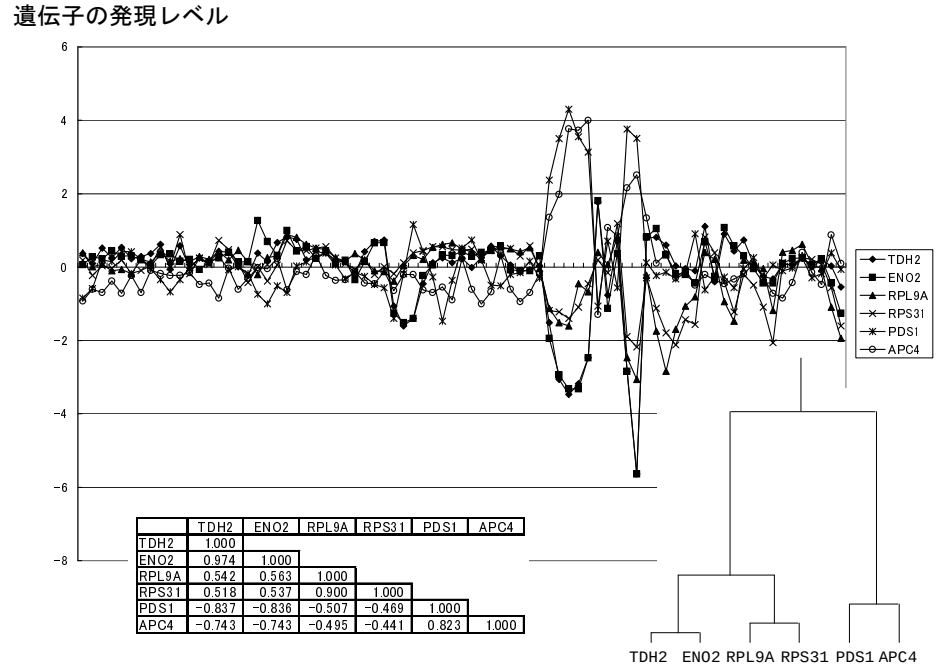
図 5.6 に酵母菌の 6 つの遺伝子の発現データの階層的クラスタリング結果を載せました。TDH2 と ENO2 は解糖に関係する酵素、RPL9A と RPS31 はリボソームのサブユニット、PDS1 と APC4 は細胞周期に関係する遺伝子です。発現相関が高いものほど枝の長さを短くしました。TDH 2 と ENO2 がまず最初にクラスタリングされ、次に RPL9A と RPS31、PDS1 と APC4 というように機能的に関連した遺伝子が最初にクラスタリングされていくのが分かります。TDH2+ENO2 のクラスターと RPL9A+RPS31 のクラスターには多少相関が見られ、PDS1+APC4 のクラスターより先に統合されていることが分かります。

5.4 遺伝子発現データの非階層的クラスタリング

非階層的クラスタリングもデータ群を似た性質のものが集まるようにするアルゴリズムです。しかしその名の通り、階層構造は取りません。非階層的クラスタリングの中でもよく使われるのが、K 平均アルゴリズム [22] です。このアルゴリズムでは、最終的にデータを k 個のクラスターに分けます。その詳細なアルゴリズムを、各要素が 2 次元データである場合を例に解説します。

各遺伝子に対して 2 つの条件で発現レベルの測定が行われた場合、データは 2 次元になります。分かりやすくするためこの発現データを図 5.7 に示すように 2 次元平面上にプロットして考えましょう。

図 5.8 では 5 個の 2 次元データ (●) がプロットされています。これを k 個のグループにクラスタリングするにはまず、参照点と呼ばれる点 (○) を k 個配置します (a)。この例では $k = 2$ の場合を示しています。そして各データが最も



	TDH2	ENO2	RPL9A	RPS31	PDS1	APC4
TDH2	1.000					
ENO2	0.974	1.000				
RPL9A	0.542	0.563	1.000			
RPS31	0.518	0.537	0.900	1.000		
PDS1	−0.837	−0.836	−0.507	−0.469	1.000	
APC4	−0.743	−0.743	−0.495	−0.441	0.823	1.000

図 5.6　酵母菌発現データの階層的クラスタリング

グラフは酵母菌の６つの遺伝子の発現パターンを表しており、横軸が発現レベルを計測した条件、縦軸が発現レベルを表す。発現データは Eisen et al. [19] より取得した。左下の表は６つの遺伝子間の発現相関を表す。右下は階層的クラスタリングの結果。相関が高かった組の枝を短く表示し、相関が低かった組の枝は長く表示した。

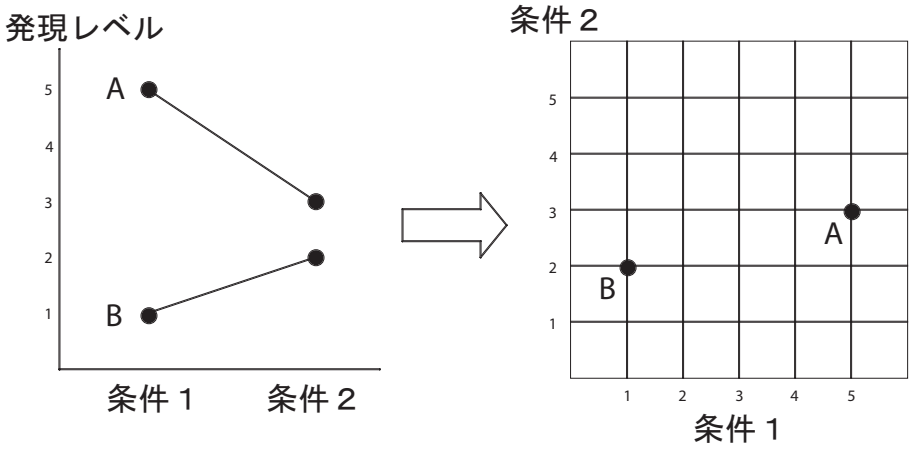


図 5.7　２つの条件で測られた発現レベルの２次元平面へのプロット

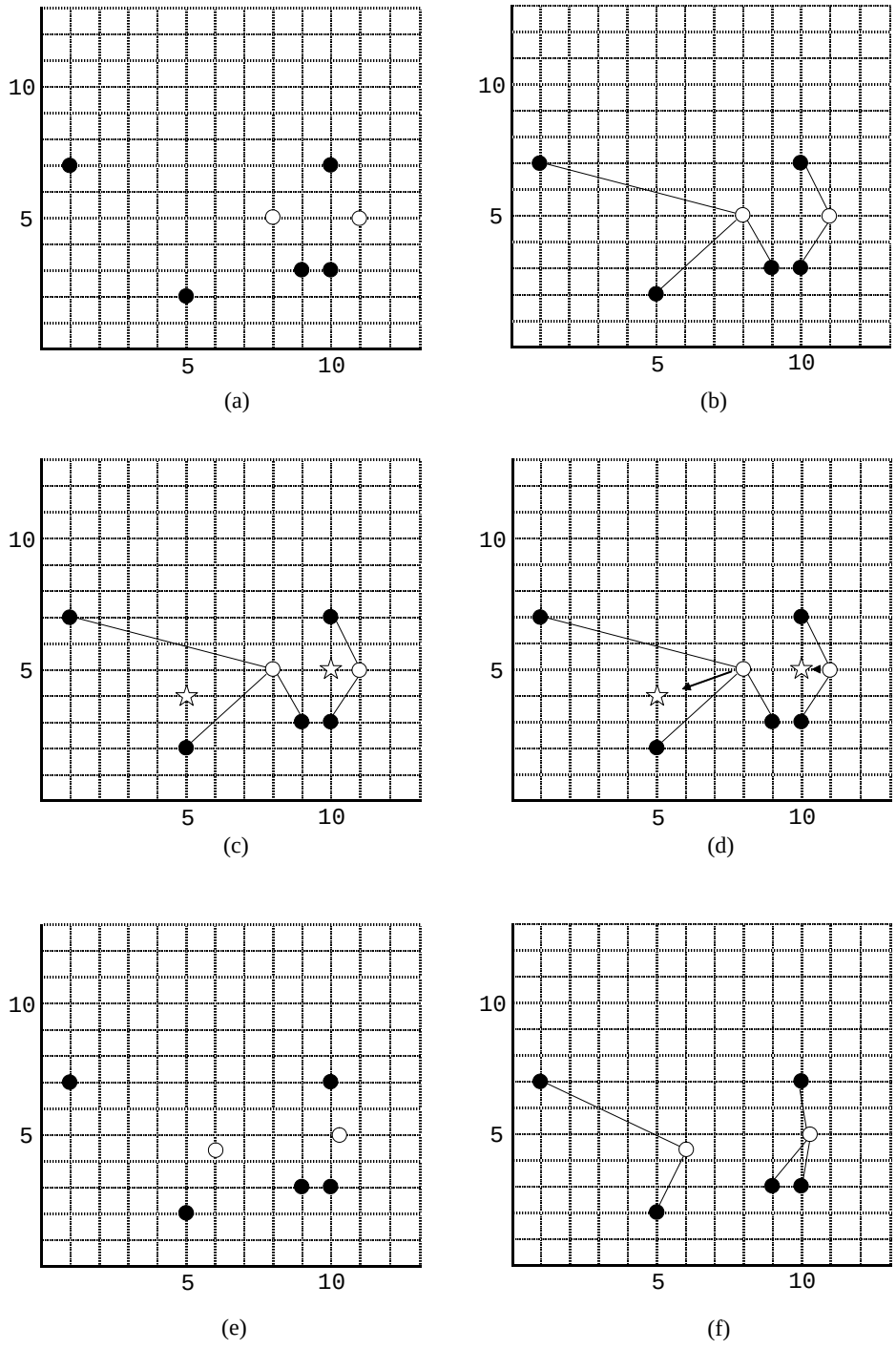


図 5.8　K 平均アルゴリズムの概要

近い参照点に属するようにします (b)。この例では距離をユークリッド距離によって測っています。図では、各点とその所属する参照点を直線で結んでいます。そして同じ参照点に属するデータは同じクラスターに属するものとみなします。

次に各クラスターに属する点の座標から重心 (☆) を求めます (c)。そして次に参照点を重心の方向へ移動させます (d, e)。この例では、参照点から重心までの直線距離の 2/3 を移動距離としています。そしてあとは (b) と同じようにまた、各データが最も近い参照点に属するようにします (f)。今回は (9,3) の点の所属が左側の参照点から右側の参照点に変わっているのが分かると思います。これを各参照点と重心が重なって収束するまで繰り返します。

まとめると、K 平均アルゴリズムは以下のように要約されます。

1. k 個の参照点を配置する。
2. 各データの所属する参照点を決める。各データの所属を最も近い参照点とする
3. 各クラスターに属するデータの重心を計算する。
4. 参照点を重心の方向へわずかに移動させる
5. 2 に戻る

以上のステップを、収束するまで繰り返します。

K 平均アルゴリズムの実装例を以下に示します。

```
/*
(1) CLUSTERS はクラスターの数 k を表す
(2) DIM はデータの次元数
(3) POINTS はデータの数
(4) RATIO は毎回の参照点移動時に、(重心の位置 - 参照点の位置) の何倍
(<1) の距離を移動させるかを表す
ともに#define で定義すること
*/

void kmeans(){
  static double ref_position[CLUSTERS][DIM];
  /* 各参照点の座標。最初の添え字は何番目の参照点かを表し、二番目の添え字は参照点の次元を表す。 */
  static double points[POINTS][DIM];
  /* 各データの座標。最初の添え字は何番目のデータかを表し、二番目の添え字は各データの次元を表す。 */
  static int points_belong[CLUSTERS][POINTS];
  /* 各データが所属する参照点。例えば i 番目のデータが k 番目の参照点
```

```
に属していた場合、points_belong[i][k] = 1 となる。そうでない場合は、
points_belong[i][k] = 0 となる。 */

  int i, j, k, iteration;

  static int next_points_belong[CLUSTERS][POINTS];
  /* 次のステップで各データが所属する参照点 */

  static double target_position[CLUSTERS][DIM];
  /* 重心の位置。最初の添え字が何番目の参照点かを表し、二番目の添え字
が各データの次元を表す */

  initialize(ref_position, points);
  /* 初期化。各データを points に格納し、参照点を ref_position に入れ
る */

  for(iteration = 0; iteration < 1000;iteration ++){ /* 1000回
のループ (収束したときに止めてもよい) */
    determine_cluster(next_points_belong, ref_position, points);
    /* 各データが次に所属する参照点を決定する */

    for(i = 0;i < CLUSTERS;i ++)
      for(j = 0;j < POINTS;j ++)
        points_belong[i][j] = next_points_belong[i][j];
    /* 各データが所属する参照点を更新 */

    calc_gravity_center(points_belong, target_position, ref_position, points);
    /* 各クラスターの重心を求め、target_position に格納する。 */

    for(i = 0;i < CLUSTERS;i ++){
      for(k = 0;k < DIM;k ++){
        ref_position[i][k] +=
           (target_position[i][k] - ref_position[i][k]) * RATIO;

      }
    } /* 参照点を重心の方向に向かって移動させる */

  }
```

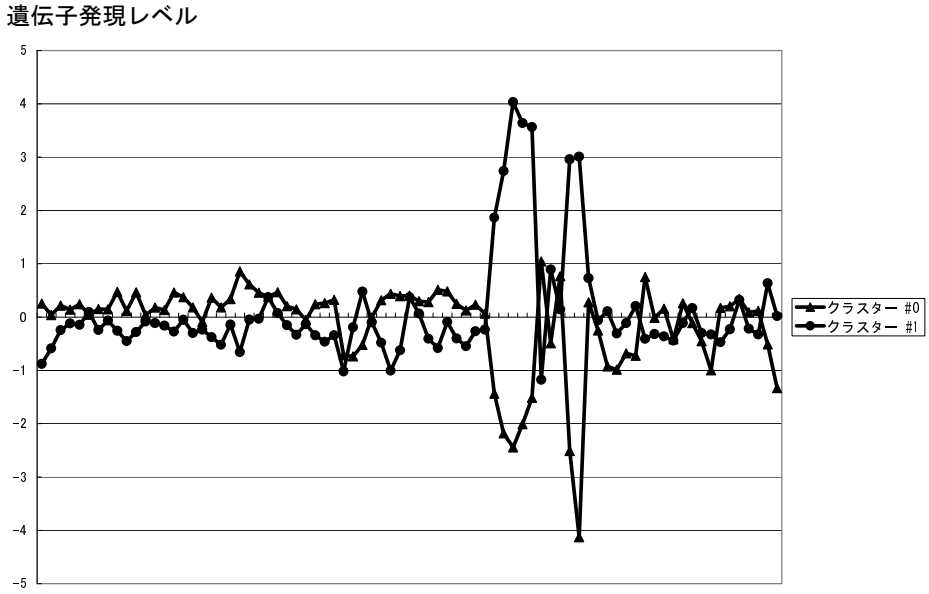


図 5.9　酵母菌発現データの K 平均アルゴリズムによるクラスタリング
図中の折れ線は各クラスターの参照点を表している。TDH2,ENO2,RPL9A,RPS31 がクラスター#0 に、PDS1 と APC4 がクラスター#1 に入っている。

}

図 5.9 に階層的クラスタリングで使用したデータセットを K 平均アルゴリズムでクラスタリングした結果を示します。この例ではユークリッド距離を類似性の指標として使いました。図には 2 つのクラスターの参照点を折れ線として示しています。$k = 2$ でクラスタリングを行っており、TDH2,ENO2,RPL9A,RPS31 がクラスター#0 に、PDS1 と APC4 がクラスター#1 に入りました。図 5.6 と比較しましょう。階層的クラスタリングでは TDH2+ENO2+RPL9A+RPS31 というクラスターが 4 回目のクラスターの統合でできましたが、それが K 平均アルゴリズムでは 1 つのクラスターにまとまっています。図 5.6 発現パターンのグラフで中央より少し右のところで発現レベルが大きく上がっていた PDS1 と APC4 が K 平均アルゴリズムで同じクラスターになり、図 5.9 の参照点を表す線が PDS1 および APC4 の発現パターンと良く似た形になっているのが分かります。また同様に TDH2,ENO2,RPL9A,RPS31 は図 5.6 で逆に発現レベルが大きく下がっていましたが、4 つとも同じクラスターになり、参照点を表す折れ線がこれらの遺伝子の発現パターンと類似しているのが分かります。

このように K 平均アルゴリズムを使うと、データを指定したグループ数 (k)

にクラスタリングできるのが分かります。

演習問題

5.1 遺伝子 1 の発現レベル $x_{11}, x_{12}, x_{13}, \cdots, x_{1n}$ と、遺伝子 2 の発現レベル $x_{21}, x_{22}, x_{23}, \cdots, x_{2n}$ が $x_{2i} = ax_{1i} + b (1 \leq i \leq n$、$a, b$ は定数) の関係にあるとき、この 2 つの遺伝子の発現レベルの相関係数は 1 になることを示しましょう。

5.2 階層的クラスタリングを行うアルゴリズムを実装し、実際の発現データを階層的クラスタリングにかけてみましょう。

5.3 5.4 のプログラムを動かすのに必要な関数 initialize, determine_cluster, calc_gravity_center, main を完成させて、K 平均アルゴリズムを実装したプログラムを動かしましょう。

第 6 章
コドンバイアスの解析

生化学情報にはコドン使用や、時系列および組織別の遺伝子発現量など、様々な多次元データが存在します。例えばコドンは終止コドンを除くと 61 種類ありますから、コドン使用は 61 次元データになりますし、各遺伝子に対して 7 つの時点での発現量を測った場合、その発現データは 7 次元データになります。

多次元情報は多くの情報を含んでいますが、要素間の関係を多次元情報から直接理解するのは困難です。人が直感的に理解できるのは、せいぜい 3 次元までででしょう。そこで本章では、コドンバイアスを題材として、多次元データを 2 次元など低い時限に集約して全体の傾向を把握する方法について説明します。

6.1　コドンバイアスとは？

mRNA の情報に基づいてタンパク質が合成されるとき、tRNA という分子が mRNA 上のコドンと呼ばれる 3 塩基で 1 組の暗号を読んで、そのコドンに対応するアミノ酸をつなげてゆきます (図 6.1)。どのコドンがどのアミノ酸をコードするかということは全生物でほぼ共通です。

さて、20 種類のアミノ酸が 61 種類のコドンによってコードされるため、あるアミノ酸をコードするコドンは 1 種類ではない場合が当然あります。例えばバリン (V) をコードするコドンは、gta, gtc, gtg, gtt の 4 種類です。このように同じアミノ酸をコードするコドンを、同義コドンと呼びます。多くの同義コドンはコドンの 3 番目だけが異なっています。

では同義コドンは均等に使われているのでしょうか？答えは否で多くの生物の場合、同義コドンは均等には使われていないことが知られています。このように同義コドンの使用の偏りのことを**コドンバイアス**と呼びます。

コドンバイアスを生じさせる主な原因は 2 つあります [23], [24]。1 つはゲノム全体にかかる変異圧です。GC 含量が高い種のゲノムには、GC 方向に偏った変異圧が作用し、AT 含量が高い種のゲノムには AT 方向に偏った変異圧が作

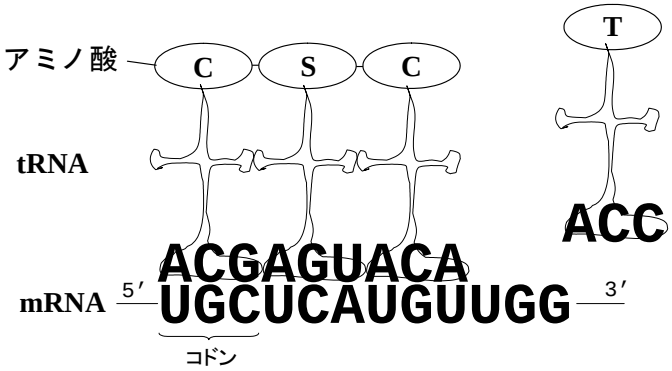


図 6.1　tRNA によるコドンの翻訳

用していると考えられています[28], [29]。すると例えばGC含有量が高い種の場合、コドンの3番目の塩基はGまたはCになりやすく、これがコドンバイアスを生じさせる原因の1つとなっています。

もう1つは翻訳効率です。あるコドンを決まったアミノ酸に翻訳可能なtRNAは1種類とは限りません。すると同義コドンの中でも、対応するtRNAの数が多いものを使った方が翻訳効率は上昇します。大腸菌の場合、リボソームタンパク質をコードするコドンはなるべく多くのtRNAに対応するようにコドンが使われていることが知られています。

種の平均的なコドンバイスの情報を利用して外来遺伝子を推測することが可能であり、また翻訳効率とコドンバイアスとの関係を利用して、コドンバイアスから逆に翻訳効率を予測する手法が開発されています[25]。このようにコドンバイアスの解析には様々な応用が考えられます。

6.2 コドンバイアスの定量化

コドンバイアスを定量化する方法はその目的によってもいくつかに分類されますが、ここでは1つ1つのコドンの偏りの程度を定量化する指標RSCU(Relative Synonymous Codon Usage)を紹介します。X_{ij}をアミノ酸iに対応するコドンjが使われた回数、n_iをアミノ酸iに対応する同義コドン数とします$(1 \leq n_i \leq 6)$。例えば、$X_{\mathrm{L\ CUU}}$はL(Leu, ロイシン)をコードするコドンCUUが使われた回数を表し、またLをコードするコドンは6種類あるので、$n_{\mathrm{L}} = 6$です。アミノ酸iに対応するコドンが使われた回数は$\sum_{j'} X_{ij'}$です。もし各同義コドンが均等に使われるとすれば、$X_{ij} = \frac{1}{n_i} \sum_{j'} X_{ij'}$となるはずです。ここでアミノ酸$i$に対応するコドン$j$のRSCUと呼ばれる指標は以下のように定義されます。

$$\mathrm{RSCU}_{ij} = \frac{X_{ij}}{\frac{1}{n_i} \sum_{j'} X_{ij'}} \tag{6.1}$$

分母が期待値であり分子が観測値となっているので、RSCUは観測されたコドンの使用回数が期待されるコドンの使用回数よりどれだけ多いか（少ないか）を表すO/E値になっていることが分かります。

例えば、Leuに対応する6つのコドンUUA, UUG, CUU, CUC, CUA, CUGの使用回数がそれぞれ12,6,18,9,12,3だったとすると、$\mathrm{RSCU}_{\mathrm{L\ UUA}} = 1.2, \mathrm{RSCU}_{\mathrm{L\ UUG}} = 0.6, \mathrm{RSCU}_{\mathrm{L\ CUU}} = 1.8, \mathrm{RSCU}_{\mathrm{L\ CUC}} = 0.9, \mathrm{RSCU}_{\mathrm{L\ CUA}} = 1.2, \mathrm{RSCU}_{\mathrm{L\ CUG}} = 0.3$となります。

表6.1に大腸菌リボソームタンパク質S20の61個のRSCU値を示します。このタンパク質に含まれているアミノ酸の数が限られているので、この表から厳密な議論はできませんが、例えばアミノ酸"T"(スレオニン)をコードするコドンとして"act"だけが使われ、他が全く使われないなど、コドン使用の偏りが見られます。

Agca	1.26	Hcac	1.33	Pcca	0.00	Taca	0.00
Agcc	0.21	Hcat	0.67	Pccc	0.00	Tacc	0.00
Agcg	0.00	Iata	0.00	Pccg	4.00	Tacg	0.00
Agct	2.53	Iatc	2.57	Pcct	0.00	Tact	4.00
Ctgc	0.00	Iatt	0.43	Qcaa	0.40	Vgta	2.00
Ctgt	0.00	Kaaa	1.14	Qcag	1.60	Vgtc	0.00
Dgac	2.00	Kaag	0.86	Raga	0.00	Vgtg	2.00
Dgat	0.00	Lcta	0.00	Ragg	0.00	Vgtt	0.00
Egaa	2.00	Lctc	0.00	Rcga	0.00	Wtgg	0.00
Egag	0.00	Lctg	6.00	Rcgc	1.71	Ytac	2.00
Fttc	1.00	Lctt	0.00	Rcgg	0.00	Ytat	0.00
Fttt	1.00	Ltta	0.00	Rcgt	4.29		
Ggga	0.00	Lttg	0.00	Sagc	1.50	Stcg	0.00
Gggc	2.00	Matg	1.00	Sagt	0.00	Stct	3.00
Gggg	0.00	Naac	1.67	Stca	1.50		
Gggt	2.00	Naat	0.33	Stcc	0.00		

表 6.1　大腸菌 30S リボソームタンパク質 S20 の RSCU

アミノ酸の 1 文字略号を大文字で、そのコドンを小文字で示す。

コドンの偏り方は生物種によって異なり、また同じ種でも遺伝子によって異なることがあります。ではどのようにすれば同じ偏り方をしているような遺伝子を集めることができるでしょうか？単に同じ偏り方をしている遺伝子を集めるだけなら、クラスタリングアルゴリズムを使うことができます。しかしそれだけではどの遺伝子がどの遺伝子とどれくらいコドンバイアスが似ているのか、直感的に知ることができません。もちろん、距離行列を作れば正確にコドンバイアスの似ている遺伝子群が分かりますが、遺伝子の数が数百になってくると、とても目で追う事はできません。

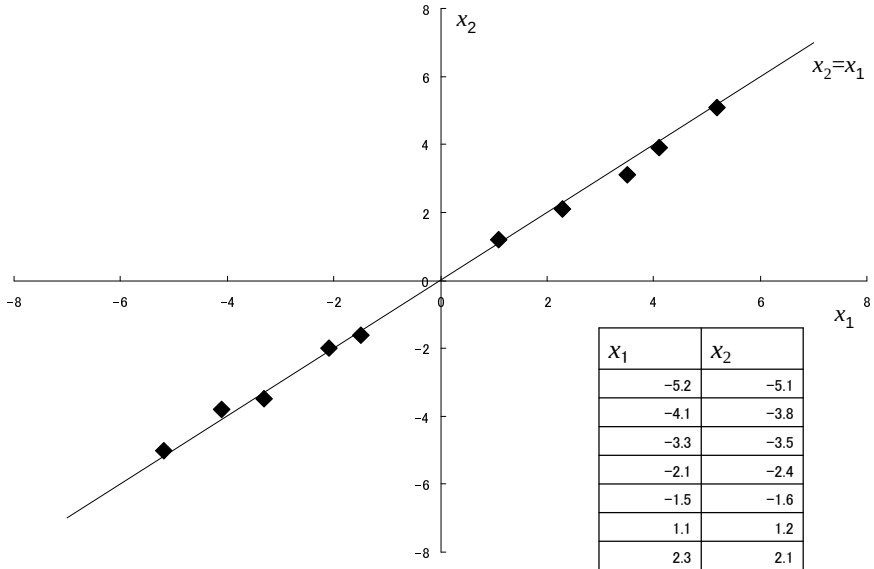
そこでコドンバイアスのような多次元のデータをより低い次元にマップして、コドンバイアスが似ている遺伝子群を俯瞰する方法を紹介します。

6.3 主成分分析

主成分分析の目的は、多次元情報を情報の損失をなるべく抑えて低い次元に移すことです[20]。まずは二次元データの主成分分析について考えてみましょう。図 6.2 の二次元座標に１０個の点があり、それらの座標は x_1 軸と、x_2 軸により表されています。ここでよく見てみると、この１０個の点に関しては、x_1 軸の値と、x_2 軸の値がほぼ同一になっていることに気づきます。実際に $x_2 = x_1$ の直線を引いてみると、１０個の点がほぼこの直線上に乗ることが分かります。そこで１０個の点がこの直線に乗っていると考えて、$x_2 = x_1$ を新しい軸とすれば、二次元の情報を一次元に圧縮できたことになります。

図 6.3 に主成分分析の原理を示します。与えられた複数の点の重心を回転部分として軸を回転させます。そして各点から軸に対して垂線を下ろし、その垂線の長さの自乗の合計が最小になるところに軸を決定します。

ではそのような軸をどのように定めればよいか、考えていきましょう。今、図 6.4 に示すように、二次元平面上に軸 x_1 と x_2 があり、点 $\mathrm{P}(x_1, x_2)$ のような点が複数存在すると考えます。問題は原点 O を通る軸 z_1 をうまく回転させて点 P からの垂線 PQ の長さの自乗 $|\overline{\mathrm{PQ}}|^2$ を最小にすることです。ここで三平方の定理より、$|\overrightarrow{\mathrm{OQ}}|^2 + |\overline{\mathrm{PQ}}|^2 = |\overrightarrow{\mathrm{OP}}|^2$ であり、また軸を回転させても $|\overrightarrow{\mathrm{OP}}|^2$ が一定のため、$|\overline{\mathrm{PQ}}|^2$ を最小にすることは、$|\overrightarrow{\mathrm{OQ}}|^2$ を最大にすることと同じです。今、$\overrightarrow{\mathrm{OQ}}$ と同じ方法を向いている単位ベクトルを $\overrightarrow{\mathrm{OR}}$ とし、R の座標を (l_1, l_2) とします。$|\overrightarrow{\mathrm{OQ}}| = \overrightarrow{\mathrm{OP}} \cdot \overrightarrow{\mathrm{OR}}$ であり、$|\overrightarrow{\mathrm{OR}}|$=1 となるので、$|\overrightarrow{\mathrm{OQ}}|^2 = (l_1 x_1 + l_2 x_2)^2$ となります。実際には複数の点についての合計を算出することになるので、1 つ目の点、２つ目の点、... をそれぞれ $(x_{11}, x_{21}), (x_{12}, x_{22}), \cdots$ として、$\sum_i (l_1 x_{1i} + l_2 x_{2i})^2$ が最大になるように l_1, l_2 を決めればよいことになります。但しここで $|\overrightarrow{\mathrm{OR}}|$=1 より、$l_1^2 + l_2^2 = 1$ という制約がつきます。従ってラグランジュの未定乗数法 (B.5 参照) を用い、λ を定数として、



x_1	x_2
−5.2	−5.1
−4.1	−3.8
−3.3	−3.5
−2.1	−2.4
−1.5	−1.6
1.1	1.2
2.3	2.1
3.5	3.1
4.1	3.9
5.2	5.1

図 6.2　二次元から一次元へ

x_1 と x_2 軸からなる座標上には 10 個の点と、直線 $x_2 = x_1$ を描いた。右下の表に 10 個の点の正確な位置を記した。

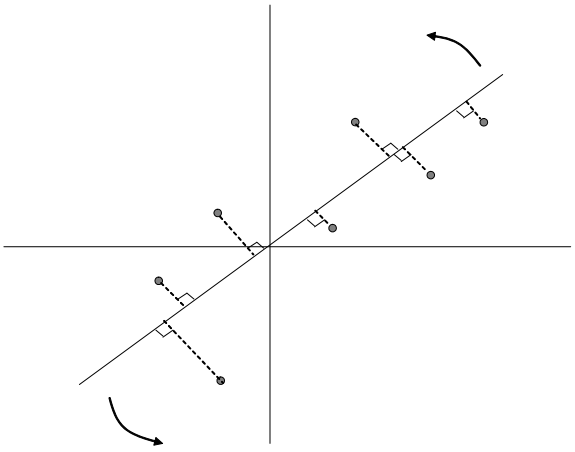


図 6.3　主成分分析の原理

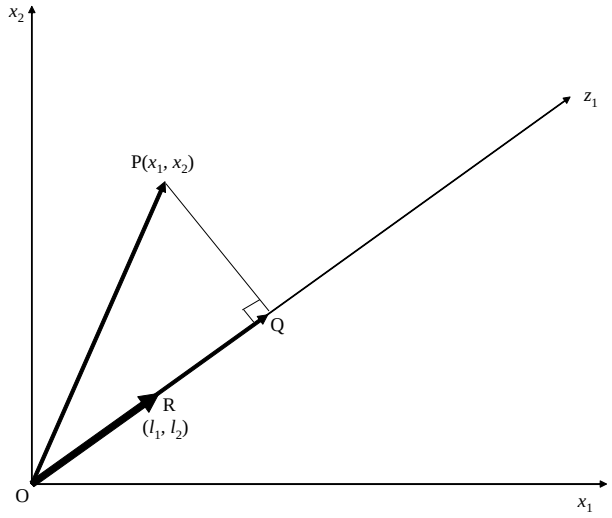


図 6.4　軸の設定

$$g(l_1, l_2) = \sum (l_1 x_{1i} + l_2 x_{2i})^2 - \lambda(l_1^2 + l_2^2 - 1) \tag{6.2}$$

を最大にすればよいことになります。そのためには式 6.2 を l_1 および l_2 で偏微分して、

$$\frac{\partial g}{\partial l_1} = \sum_i 2(l_1 x_{1i} + l_2 x_{2i}) x_{1i} - 2\lambda l_1 = 0 \tag{6.3}$$

$$\frac{\partial g}{\partial l_2} = \sum_i 2(l_1 x_{1i} + l_2 x_{2i}) x_{2i} - 2\lambda l_2 = 0 \tag{6.4}$$

を解きます。軸 x_1 の値の分散は $s_{11} = \frac{1}{m}\sum_{i=1}^{m}(x_{1i} - \overline{x_1})^2$ であり、軸 x_2 の値の分散は $s_{22} = \frac{1}{m}\sum_{i=1}^{m}(x_{2i} - \overline{x_2})^2$ となります。但し $\overline{x_1}, \overline{x_2}$ はそれぞれ軸 x_1 の値の平均および軸 x_2 の値の平均です。また軸 x_1 と軸 x_2 の値の共分散は、$s_{12} = s_{21} = \frac{1}{m}\sum_{i=1}^{n}(x_{1i} - \overline{x_1})(x_{2i} - \overline{x_2})$ です。対象となる m 個の点の重心がちょうど原点になるようにしておけば、$\overline{x_1} = 0, \overline{x_2} = 0$ となります。すると式 6.3、6.4 は

$$\frac{1}{m}\begin{pmatrix} s_{11} & s_{21} \\ s_{12} & s_{22} \end{pmatrix}\begin{pmatrix} l_1 \\ l_2 \end{pmatrix} = \lambda \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}$$

と書き表すことができます。s_{ij} を m 倍したものを改めて s_{ij} に定義し直せば、

$$\begin{pmatrix} s_{11} & s_{21} \\ s_{12} & s_{22} \end{pmatrix}\begin{pmatrix} l_1 \\ l_2 \end{pmatrix} = \lambda \begin{pmatrix} l_1 \\ l_2 \end{pmatrix} \tag{6.5}$$

とより単純な形になります。ここで

$$X = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1m} & x_{2m} \end{pmatrix}, L = \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}$$

とおけば式 6.5 は

$$X^t X L = \lambda L \tag{6.6}$$

と書き直すことができます。6.6 の解となる λ を $X^t X$ の固有値、L を固有ベクトルと呼びます[26]。

さて我々が知りたいのは、各点 (x_1, x_2) が z_1 軸上のどの位置に対応するかということです。i 番目の点の z_1 軸上の座標 z_{1i} は $|\overrightarrow{\mathrm{OQ}}| = \overrightarrow{\mathrm{OP}} \cdot \overrightarrow{\mathrm{OR}}$ より、以下の式により求めることができます。

$$z_{1i} = l_1 x_{1i} + l_2 x_{2i} = (x_{1i} \quad x_{2i}) \begin{pmatrix} l_1 \\ l_2 \end{pmatrix}$$

ところで式 6.6 の解は 1 つとは限りません。L の解[*1)]は最大で 2 つ考えられ、

*1) 互いに線形独立な解

それに対応する λ も 2 つとなります。そこで L の 2 つの解を $L_1 = (l_{11}, l_{21})^t$、$L_2 = (l_{12}, l_{22})^t$、対応する λ の値を λ_1, λ_2 としましょう。どちらがより $|\overrightarrow{\mathrm{OQ}}|^2$ を大きくするでしょうか？$|\overrightarrow{\mathrm{OQ}}|^2$ は、

$$|\overrightarrow{\mathrm{OQ}}|^2 = \sum_i (l_1 x_{1i} + l_2 x_{2i})^2 = l_1^2 \sum_i x_{1i}^2 + 2 l_1 l_2 \sum_i x_{1i} x_{2i} + l_2^2 \sum_i x_{2i}^2$$

と表すことができますが、ここで先ほどと同じように、$s_{11} = \sum_i x_{1i}^2$、$s_{22} = \sum_i x_{2i}^2$ を使うと、

$$|\overrightarrow{\mathrm{OQ}}|^2 = l_1^2 s_{11} + 2 l_1 l_2 s_{12} + l_2^2 s_{22} = l_1 (l_1 s_{11} + l_2 s_{12}) + l_2 (l_1 s_{12} + l_2 s_{22})$$

となります。さらに式 6.5 を使うと $|\overrightarrow{\mathrm{OQ}}|^2$ は、

$$|\overrightarrow{\mathrm{OQ}}|^2 = l_1 \cdot \lambda l_1 + l_2 \cdot \lambda l_2 = \lambda (l_1^2 + l_2^2) = \lambda$$

より $|\overrightarrow{\mathrm{OQ}}|^2 = \lambda$ となるので、大きい方の λ を用いた方が $|\overrightarrow{\mathrm{OQ}}|^2 = \lambda$ は大きくなります。

$\lambda_1 \geq \lambda_2$ として、式 6.6 は以下のように表現することができます。

$$X^t X (L_1 L_2) = (L_1 L_2) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

各点の新しい軸上での座標は、

$$X(L_1, L_2) = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1m} & x_{2m} \end{pmatrix} \begin{pmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11} x_{11} + l_{21} x_{21} & l_{12} x_{11} + l_{22} x_{21} \\ l_{11} x_{12} + l_{21} x_{22} & l_{12} x_{12} + l_{22} x_{22} \\ \vdots & \vdots \\ l_{11} x_{1m} + l_{21} x_{2m} & l_{12} x_{1m} + l_{22} x_{2m} \end{pmatrix}$$

となります。二次元データを一次元にマッピングしたい場合は、この行列の一列目を抽出すればいいことになります。

次にこれを n 次元データに拡張してみましょう。今、m 個の n 次元データ $f_1, f_2, \cdots, f_m$ を以下のように行列 F を用いて表します。

$$F = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{m-1} \\ f_m \end{pmatrix} = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1 \cdot n-1} & f_{1n} \\ f_{21} & f_{22} & & & f_{2n} \\ \vdots & & \ddots & & \\ f_{m-1 \cdot 1} & & & f_{m-1 \cdot n-1} & f_{m-1 \cdot n} \\ f_{m1} & & & f_{m \cdot n-1} & f_{mn} \end{pmatrix}$$

但し f_{ij} は i 番目のデータの j 次元目の値を表します。次に各列の平均を以下の式のように計算します。

$$\overline{f_{\cdot j}} = \frac{1}{m}\sum_{k=1}^{m} f_{kj}$$

次に元の行列 F から各列の平均を計算した行列 $\overline{F}$ を引き、F_c とします。

$$F_c = F - \overline{F} = F - \begin{pmatrix} \overline{f_{\cdot 1}} & \overline{f_{\cdot 2}} & \cdots & \overline{f_{\cdot n-1}} & \overline{f_{\cdot n}} \\ \overline{f_{\cdot 1}} & \overline{f_{\cdot 2}} & \cdots & \overline{f_{\cdot n-1}} & \overline{f_{\cdot n}} \\ \vdots & & & & \\ \overline{f_{\cdot 1}} & \overline{f_{\cdot 2}} & \cdots & \overline{f_{\cdot n-1}} & \overline{f_{\cdot n}} \\ \overline{f_{\cdot 1}} & \overline{f_{\cdot 2}} & \cdots & \overline{f_{\cdot n-1}} & \overline{f_{\cdot n}} \end{pmatrix}$$

さて、l_1 列目と l_2 列目の間の分散 ($l_1 = l_2$ のとき) および共分散 ($l_1 \neq l_2$ のとき) を $s_{l_1 l_2}$ で表すと、

$$s_{l_1 l_2} = \frac{1}{m}\sum_{k=1}^{m}(f_{kl_1} - \overline{f_{\cdot l_1}})(f_{kl_2} - \overline{f_{\cdot l_2}})$$

となるので、

$$F_c^t F_c = m \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1\cdot n-1} & s_{1n} \\ s_{21} & s_{22} & & & s_{2n} \\ \vdots & & \ddots & & \\ s_{n-1\cdot 1} & & & s_{n-1\cdot n-1} & s_{n-1\cdot n} \\ s_{n1} & & & s_{n\cdot n-1} & s_{nn} \end{pmatrix}$$

が成立します。さて、

$$F_c^t F_c p_i = \lambda_i p_i, i = 1, 2, \cdots, n$$

を満たすような n 次元ベクトル p_i(固有ベクトル) およびそれに対応する λ_i(固有値) を求めると $\lambda_i \geq \lambda_{i+1}, 1 \leq i < n$ として、

$$F_c^t F_c = P\Lambda P^{-1} = (p_1, p_2, \cdots, p_{n-1}, p_n)\Lambda(p_1, p_2, \cdots, p_{n-1}, p_n)^t$$

$$= \begin{pmatrix} p_{11} & p_{21} & \cdots & p_{n1} \\ p_{12} & p_{22} & & p_{n2} \\ \vdots & & \ddots & \\ p_{1n} & & & p_{nn} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & \\ \vdots & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix} \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & & p_{2n} \\ \vdots & & \ddots & \\ p_{n1} & & & p_{nn} \end{pmatrix}$$

のように間に対角行列を作ることができます。

データ $f_1, f_2, \cdots, f_m$ の新しい座標 $z_1, z_2, \cdots, z_m$ は以下のように表すことができます。

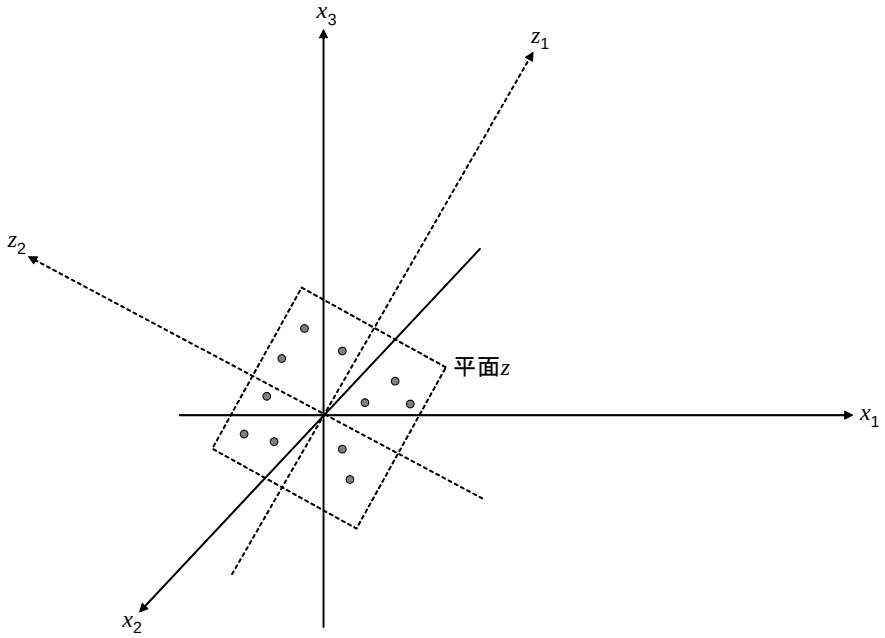


図 6.5　主成分分析による 3 次元データの 2 次元へのマッピング

$$Z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} & \cdots & z_{1n} \\ z_{21} & z_{22} & & \\ \vdots & & \ddots & \\ z_{m1} & & & z_{mn} \end{pmatrix} = F_c P = (F - \overline{F})P$$

従って、i 番目のデータの j 次元目の値は、

$$z_{ij} = \sum_{k=1}^{n} (f_{ik} - \overline{f_{\cdot k}}) p_{jk}$$

となります。従って n 次元データ $f_1, f_2, \cdots, f_m$ を一次元にマッピングすると、それぞれの一次元座標は $z_{11}, z_{21}, \cdots, z_{m1}$ となります。

さてこれまでは、多次元データを一次元にマッピングすることを考えてきました。しかしその多次元データがある直線上周辺に分布していない場合、一次元へのマッピングは多くの情報を失うことになるため、必ずしも効果的とは言えません。一方でこれらデータがある平面上に分布している場合、二次元平面へのマッピングが効果的と言えます。図 6.5 では、x_1, x_2, x_3 の 3 つの軸からなる三次元空間内に三次元データに対応する点がプロットされています。そしてプロットされている点は平面 z 上に載っていることが分かります。従って、三次元上の各点は平面 z 上の軸 z_1 と z_2 を使って二次元データとして表すことができます。通常、n 次元データ $f_1, f_2, \cdots, f_m$ を二次元にマッピングすると、それぞれの二次元座標

```
PCA.calc <- function(F){
   PCA.res <- prcomp(F)
   P <- PCA.res$rotation
   Z <- PCA.res$x
   Fc <- t(t(F) - apply(F, 2, mean))
   L <- t(P) %*% t(Fc) %*% Fc %*% P
   return(list(F=F, P=P, Z=Z, Fc=Fc, L=L))
}
```

図 6.6　主成分分析を行う R 言語のプログラム
主成分分析を行いたい行列 F を `PCA.calc(F)` のように関数に渡すと、
本文中で説明している行列 P,Z,Fc,Λ を P,Z,Fc,L のリストとして返す。

は Z の 1 次元目と 2 次元目の値を用いて、$(z_{11}, z_{12}), (z_{21}, z_{22}), \cdots, (z_{m1}, z_{m2})$ となります。

多次元データをより低い k 次元にマッピングしたときに、どれくらい元のデータを反映しているかの指標として寄与率 (累積寄与率) [20] があり、$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$ で定義されます*2)。寄与率が高ければ高いほど、元のデータの情報の損失を抑えて低次元にマッピングしていることになります。例えば二次元データを一次元データにマッピングするとき、元の二次元データが完全に直線上に乗っていれば、寄与率は 100%となります。

次元の数が多くなると、これらを手計算で行うのは非常に困難です。そこで R 言語 [27] と呼ばれる統計処理言語を使用すると、これらの行列計算を簡単に行うことができます。図 6.6 に主成分分析を行うプログラムを掲載しておきます。

6.4　コドンバイアスの主成分分析

それでは実際にコドンバイアスを主成分分析にかけてみましょう。まず行列 F を構築しなければなりません。図 6.7 のようにまず各遺伝子の 61 個のコドンに対して、RSCU 値を求めます。そして行を各遺伝子に、列を各コドンに対応させるように、RSCU 値を表す行列 F を作ります。f_{ij} は i 番目の遺伝子の j 番目 $(1 \leq j \leq 61)$ の RSCU 値になります。

行列 F に対して Z を求め、各遺伝子 i の (z_{i1}, z_{i2}) をプロットしたのが図 6.8 です。各点が各遺伝子のコドン使用を 2 次元に圧縮したものを表します。リボソームタンパク質と tRNA シンセターゼの遺伝子は全体の分布に比べると偏った場所に位置しているのが分かると思います。リボソームタンパク質や tRNA

*2)　$\sum_{i=1}^{n} \lambda_i = m \sum_{i=1}^{n} s_{ii}$ が成立する。詳しくは文献 [20], [26] 参照。

Gene	Agca	Agcc	Agcg	…	Wtgg	Ytac	Ytat
thrL	0.00	0.00	4.00	…	0.00	0.00	0.00
thrA	0.62	1.58	1.14	…	1.00	0.80	1.20
thrB	0.65	0.97	1.84	…	1.00	0.75	1.25
thrC	0.43	0.68	2.04	…	1.00	1.11	0.89
:	:	:	:		:	:	:
:	:	:	:		:	:	:
yjjY	2.67	0.00	0.00	…	0.00	0.00	0.00
lasT	1.60	0.80	1.20	…	1.00	0.67	1.33

$$\longrightarrow F = \begin{pmatrix} 0.00 & 0.00 & 4.00 & \cdots & 0.00 & 0.00 & 0.00 \\ 0.62 & 1.58 & 1.14 & \cdots & 1.00 & 0.80 & 1.20 \\ 0.65 & 0.97 & 1.84 & \cdots & 1.00 & 0.75 & 1.25 \\ 0.43 & 0.68 & 2.04 & \cdots & 1.00 & 1.11 & 0.89 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 2.67 & 0.00 & 0.00 & \cdots & 0.00 & 0.00 & 0.00 \\ 1.60 & 0.80 & 1.20 & \cdots & 1.00 & 0.67 & 1.33 \end{pmatrix}$$

図 6.7　行列 F の構築

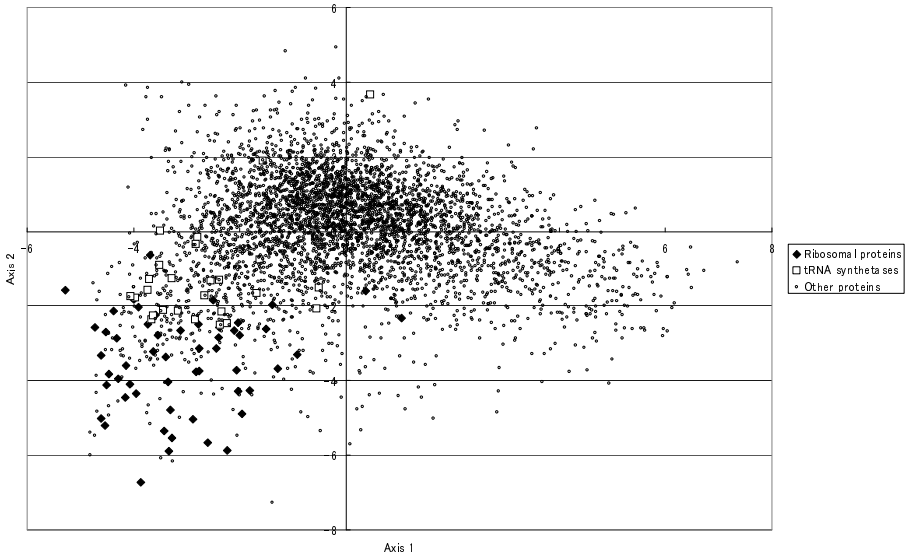


図 6.8　大腸菌のコドン使用の主成分分析結果

シンセターゼはタンパク質の合成に必須であり、多くの細胞で多量に発現しています。同義コドンのうち、対応する tRNA が細胞内に多く存在するものを使用すると、タンパク質の合成効率が良くなることが知られています[25]。従って、高発現遺伝子では細胞内に豊富にある tRNA に対応するコドンを特に使用する傾向があると考えられ、これが分布上の偏りとなって現れていると考えられます。

6.5 対応分析

対応分析*3)も主成分分析と同様、多次元データを低い次元にマッピングします。対応分析ではさらに行と列を同じ座標にマッピングすることにより、行の項目と列の項目との関係を把握することができます[31]。

それでは対応分析ではどのような計算が行われるのか、説明していきます。まず対象となる全ての要素が分類基準 A において、クラス 1〜m のいずれかのクラスに属し、また分類基準 B において、クラス 1〜n までのいずれかのクラスに属するとしましょう。コドンバイアスの例では A は遺伝子、B はコドンということになるでしょう。分類基準 A ではクラス i に属し、分類基準 B ではクラス j に属する要素の数を f_{ij} と表します。クラスごとの要素の数は以下のように $m \times n$ の行列 F でまとめて表すことができます。

$$F = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1\cdot n-1} & f_{1n} \\ f_{21} & f_{22} & & & f_{2n} \\ \vdots & & \ddots & & \\ f_{m-1\cdot 1} & & & f_{m-1\cdot n-1} & f_{m-1\cdot n} \\ f_{m1} & & & f_{m\cdot n-1} & f_{mn} \end{pmatrix}$$

ここで、$f_{i\cdot} = \sum_{l=1}^{n} f_{il}, f_{\cdot j} = \sum_{k=1}^{m} f_{kj}$ として、行列 S, C を以下のように定義します。

$$S^{-\frac{1}{2}} = \begin{pmatrix} \frac{1}{\sqrt{f_{1\cdot}}} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\sqrt{f_{2\cdot}}} & & & \\ \vdots & & \ddots & & \\ 0 & & & \frac{1}{\sqrt{f_{m-1\cdot}}} & \\ 0 & & & & \frac{1}{\sqrt{f_{m\cdot}}} \end{pmatrix}$$

*3) 数学的構造は、数量化　類と呼ばれる手法と同一。

$$C^{-\frac{1}{2}} = \begin{pmatrix} \frac{1}{\sqrt{f_{\cdot 1}}} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\sqrt{f_{\cdot 2}}} & & & \\ \vdots & & \ddots & & \\ 0 & & & \frac{1}{\sqrt{f_{\cdot n-1}}} & \\ 0 & & & & \frac{1}{\sqrt{f_{\cdot n}}} \end{pmatrix}$$

また行列 F, S, C を使って行列 H を以下のように定義します。

$$H = S^{-\frac{1}{2}} F C^{-\frac{1}{2}} = \begin{pmatrix} \frac{f_{11}}{\sqrt{f_{1\cdot} f_{\cdot 1}}} & \frac{f_{12}}{\sqrt{f_{1\cdot} f_{\cdot 2}}} & \cdots & \cdots & \frac{f_{1n}}{\sqrt{f_{1\cdot} f_{\cdot n}}} \\ \frac{f_{21}}{\sqrt{f_{2\cdot} f_{\cdot 1}}} & \frac{f_{22}}{\sqrt{f_{2\cdot} f_{\cdot 2}}} & & & \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ \frac{f_{m1}}{\sqrt{f_{m\cdot} f_{\cdot 1}}} & & & & \frac{f_{mn}}{\sqrt{f_{m\cdot} f_{\cdot n}}} \end{pmatrix} \tag{6.7}$$

ここで以下のように式 6.10、6.11、6.13 を満たすように行列 U, V を決めます。$m > n$、$d_{ii} \geq d_{i+1\cdot i+1}, 1 \leq i < m$ として、

$$HH^t = S^{-\frac{1}{2}} F C^{-1} F^t S^{-\frac{1}{2}} = U D^2 U^t = (u_1, u_2, \cdots, u_m) D^2 (u_1, u_2, \cdots, u_m)^t$$

$$= \begin{pmatrix} u_{11} & \cdots & u_{m1} \\ \vdots & \ddots & \\ u_{1m} & & u_{mm} \end{pmatrix} \begin{pmatrix} d_{11}^2 & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & d_{mm}^2 \end{pmatrix} \begin{pmatrix} u_{11} & \cdots & u_{1m} \\ \vdots & \ddots & \\ u_{m1} & & u_{mm} \end{pmatrix} \tag{6.8}$$

但し u_i は m 次元ベクトル、D は $d_{ii}(1 \leq i \leq m)$ を対角要素とする $m \times m$ 対角行列で、$d_{11} = 1$ となります (B.6 参照)。また、

$$u_i^t u_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{6.9}$$

が成立するようにします。こうすると実際は、$n < i \leq m$ に対して $d_{ii} = 0$ となるので式 6.8 は、

$$HH^t = (u_1, u_2, \cdots, u_n) \begin{pmatrix} d_{11}^2 & 0 & \cdots & 0 & 0 \\ 0 & d_{22}^2 & & & \\ \vdots & & \ddots & & \\ 0 & & & d_{n-1\cdot n-1}^2 & \\ 0 & & & & d_{nn}^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} \tag{6.10}$$

と表すことができます。同様にして、

$$H^t H = V D^2 V^t$$

$$= (v_1, v_2, \cdots, v_n) \begin{pmatrix} d_{11}^2 & 0 & \cdots & 0 & 0 \\ 0 & d_{22}^2 & & & \\ \vdots & & \ddots & & \\ 0 & & & d_{n-1\cdot n-1}^2 & \\ 0 & & & & d_{nn}^2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{pmatrix} \quad (6.11)$$

但し、v_i は n 次元ベクトルです。また

$$v_i^t v_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad (6.12)$$

が成立するようにします。最後に

$$H = UDV^t \quad (6.13)$$

を満たすようにします。ちなみに H に対してこのような U, D, V を求めることを、H の特異値分解と呼びます [31]。ここまでできればいよいよ最後の低次元へのマッピングに入ります。まず $f_{\cdot\cdot} = \sum_{k=1}^{m} \sum_{l=1}^{n} f_{kl}$ として、

$$X = (x_1, \cdots, x_n) = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n-1\cdot 1} & x_{n1} \\ x_{12} & x_{22} & & & x_{n2} \\ \vdots & & \ddots & & \\ x_{1\cdot m-1} & & & x_{n-1\cdot m-1} & x_{n\cdot m-1} \\ x_{1m} & & & x_{n-1\cdot m} & x_{nm} \end{pmatrix}$$

$$= \sqrt{f_{\cdot\cdot}} S^{-\frac{1}{2}} U = \sqrt{f_{\cdot\cdot}} \begin{pmatrix} \frac{1}{\sqrt{f_{1\cdot}}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{f_{2\cdot}}} & & \\ \vdots & & \ddots & \\ 0 & & & \frac{1}{\sqrt{f_{m\cdot}}} \end{pmatrix} (u_1, \cdots, u_n) \quad (6.14)$$

を求めます。ここで x_k は m 次元ベクトルです。分類基準 A のクラス i は k 次元目では x_{ki} という値を持つことになります。但し、$x_1 = (1, 1, 1, \cdots, 1)^t$ となるので、通常 x_1 は考慮しません (B.6 節参照)。つまり例えば、クラス i を 2 次元データにマッピングすると、(x_{2i}, x_{3i}) となります。

次に

$$Y = (y_1, \cdots, y_n) = \begin{pmatrix} y_{11} & y_{21} & \cdots & y_{n-1\cdot 1} & y_{n1} \\ y_{12} & y_{22} & & & y_{n2} \\ \vdots & & \ddots & & \\ y_{1\cdot n-1} & & & y_{n-1\cdot n-1} & y_{n\cdot n-1} \\ y_{1n} & & & y_{n-1\cdot n} & y_{nn} \end{pmatrix}$$

```
CA.calc <- function(F){
   f.. <- sum(F)
   S <- diag(apply(F, 1, sum))
   C <- diag(apply(F, 2, sum))
   Srsr <- diag(apply(F, 1, sum)^(-1/2))
   Crsr <- diag(apply(F, 2, sum)^(-1/2))
   H <- Srsr %*% F %*% Crsr
   SVD.res <- svd(H)
   U <- SVD.res$u
   D <- diag(SVD.res$d)
   V <- SVD.res$v
   if(U[1,1] < 0){ U <- -U; V <- -V }
   X <- f..^(1/2) * Srsr %*% U
   Y <- f..^(1/2) * Crsr %*% V
   return(list(F=F, H=H, X=X, Y=Y, S=S, C=C, U=U, D=D, V=V))
}
```

図 6.9　対応分析を行う R 言語のプログラム

対応分析を行いたい行列 F を `CA.calc(F)` のように関数に渡すと、本文中で説明している行列 H,X,Y,S,C,U,D,V を H,X,Y,S,C,U,D,V のリストとして返す。

$$= \sqrt{f_{..}} C^{-\frac{1}{2}} V = \sqrt{f_{..}} \begin{pmatrix} \frac{1}{\sqrt{f_{\cdot 1}}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{f_{\cdot 2}}} & & \\ \vdots & & \ddots & \\ 0 & & & \frac{1}{\sqrt{f_{\cdot m}}} \end{pmatrix} (v_1, \cdots, v_n) \quad (6.15)$$

を求めます。但し、y_k は m 次元ベクトルです。分類基準 B のクラス j は k 次元目では y_{kj} という値を持つことになります。但し、$y_1 = (1, 1, 1, \cdots, 1)^t$ となるので、通常 y_1 も考慮しません (B.6 節参照)。この場合も例えばクラス j を 2 次元データにマッピングすると、(y_{2j}, y_{3j}) となります。

図 6.9 に対応分析を行う R 言語のプログラムを掲載しておきます。

さて x_k と y_k にはどのような関係があるか調べていきましょう。式 6.14、6.15 を移項して 6.13 に代入すると、

$$H = UDV^t = \frac{1}{f_{..}} S^{\frac{1}{2}} X D Y^t C^{\frac{1}{2}}$$

となります。すると式 6.7 より、

$$F = S^{\frac{1}{2}} H C^{\frac{1}{2}} = \frac{1}{f_{..}} S X D Y^t C$$

が成立します。$x_1 = (1, 1, 1, \cdots, 1)^t, y_1 = (1, 1, 1, \cdots, 1)^t, d_{11} = 1$ であることに注意すると、F の各要素 f_{ij} は

$$f_{ij} = \frac{f_{i\cdot} f_{\cdot j}}{f_{..}} \sum_{k=1}^{n} d_{kk} x_{ki} y_{kj}$$

$$= \frac{f_{i\cdot} f_{\cdot j}}{f_{..}} + \frac{f_{i\cdot} f_{\cdot j}}{f_{..}} d_{22} x_{2i} y_{2j} + \frac{f_{i\cdot} f_{\cdot j}}{f_{..}} d_{33} x_{3i} y_{3j} + \cdots + \frac{f_{i\cdot} f_{\cdot j}}{f_{..}} d_{nn} x_{ni} y_{nj} \quad (6.16)$$

となります。式 6.16 は d_{kk}, x_{ki}, y_{kj} があれば f_{ij} を復元できることを意味します。また d_{22} や d_{33} が大きく、これに比べて $d_{kk}, k > 3$ が小さければ、$x_{2k}, x_{3k}, y_{2k}, y_{3k}, d_{22}, d_{33}$ だけでも f_{ij} をほぼ復元できることを意味します。この場合、情報の欠落を抑えて、各データを 2 次元にマッピングできます。

なお、d_{kk} と χ^2 値の間には、

$$f_{..} \sum_{k=2}^{n} d_{kk} = \chi^2$$

の関係が成立しています [31]。

式 6.8 を式 6.14 を用いて変形すると、

$$\frac{1}{\sqrt{f_{..}}} S^{-\frac{1}{2}} F C^{-1} F^t S^{-\frac{1}{2}} (S^{\frac{1}{2}} x_i) = \frac{1}{\sqrt{f_{..}}} d_{ii}^2 (S^{\frac{1}{2}} x_i) \quad (6.17)$$

となります。

$M = S^{-\frac{1}{2}} F C^{-1} F^t S^{-\frac{1}{2}}$ とおいて、式 6.17 の両辺を $\sqrt{f_{ii}}$ 倍すれば、

$$M S^{\frac{1}{2}} x_i = d_{ii}^2 S^{\frac{1}{2}} x_i \quad (6.18)$$

M が対称行列であることに注意すると、x_k、x_l、d_{kk}^2、d_{ll}^2 に対し、

$$(M S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l = (d_{kk}^2 S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l = d_{kk}^2 (S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l \quad (6.19)$$

$$(S^{\frac{1}{2}} x_k)^t M S^{\frac{1}{2}} x_l = (S^{\frac{1}{2}} x_k)^t d_{ll}^2 S^{\frac{1}{2}} x_l = d_{ll}^2 (S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l \quad (6.20)$$

式 6.19 から 6.20 を引いて、

$$(d_{kk}^2 - d_{ll}^2)(S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l = 0$$

従って、$(d_{kk}^2 \neq d_{ll}^2)$ なら

$$(S^{\frac{1}{2}} x_k)^t S^{\frac{1}{2}} x_l = 0 \quad (6.21)$$

となります。従って $k > 1$ に対し、

$$S^{\frac{1}{2}} x_1 = (\sqrt{f_{1\cdot}}, \cdots, \sqrt{f_{m\cdot}})^t, \qquad S^{\frac{1}{2}} x_k = (\sqrt{f_{1\cdot}} x_{k1}, \cdots, \sqrt{f_{m\cdot}} x_{km})^t$$

$$(S^{\frac{1}{2}} x_1)^t S^{\frac{1}{2}} x_k = 0$$

が成立するので、

$$\sum_{l=1}^{m} f_{l\cdot} x_{kl} = 0 \tag{6.22}$$

となり、結局 x_k のベクトルの要素の平均は 0 となります。同様のことが y についても成立します。

次に x_k のベクトルの要素の分散について考えてみると、x_{ij} の値を持つ要素が f_{ij} 個あるので、式 6.9 に注意すると分散は、

$$\frac{1}{f_{\cdot\cdot}} x_i^t S x_i = \frac{1}{f_{\cdot\cdot}} u_i^t S^{-\frac{1}{2}} \sqrt{f_{\cdot\cdot}} S \sqrt{f_{\cdot\cdot}} S^{-\frac{1}{2}} u_i = 1$$

となります。同様の計算が y_k についても成立します。

さらに B.6 節で説明するように、対応分析では x_{ik} と y_{ik} の相関が最大になるようにクラス i に対する x_{ik} とクラス j に対する y_{ik} が決められます。

まとめると、

- x_k、y_k のベクトルの要素の平均は 0 になる
- x_k、y_k のベクトルの要素の分散は 1 になる
- x_k、y_k の相関係数がなるべく高くなるように x_k、y_k が決められる

となります。x_k, y_k の平均と分散が同じで、相関が高いということは、x_k と y_k を同一尺度で考えることができることを意味します。そして、f_{ij} が十分多ければ、ある次元 k において、x_{ki} は y_{kj} に近い値をとるようになります。これが対応分析の大きな特徴で、行と列を同一の尺度にマッピングすることができるのです。

6.6 コドンバイアスの対応分析

それでは大腸菌のコドン使用を対応分析にかけてみましょう。行列 F の作り方は主成分分析のときと同じです (図 6.7)。対応分析では、行列 F に対する X、Y が求められます。各遺伝子 i の (x_{2i}, x_{3i}) をプロットしたものが図 6.10 です。対応分析では同時に各コドン j の (y_{2j}, y_{3j}) も同じグラフにプロットすることができます。ある遺伝子とあるコドンの位置が近い場合、その遺伝子の位置がそのコドンの使用頻度に深く関係していることを表します。

図 6.10 を見ると、主成分分析を用いたときの結果と同じように、リボソームタンパク、tRNA シンセターゼが全体の分布から偏った位置にあることが分かります。横軸上ではリボソームタンパク質や tRNA シンセターゼが右の方に偏っています。

そしてコドンとの関係に注目しましょう。アルギニン (R) をコードする 6 つのコドン (aga,agg,cga,cgc,cgg,cgt) の分布を見ると、cgc、cgt は横軸上でリボソームタンパク質や tRNA シンセターゼと同様、比較的右に位置しているの

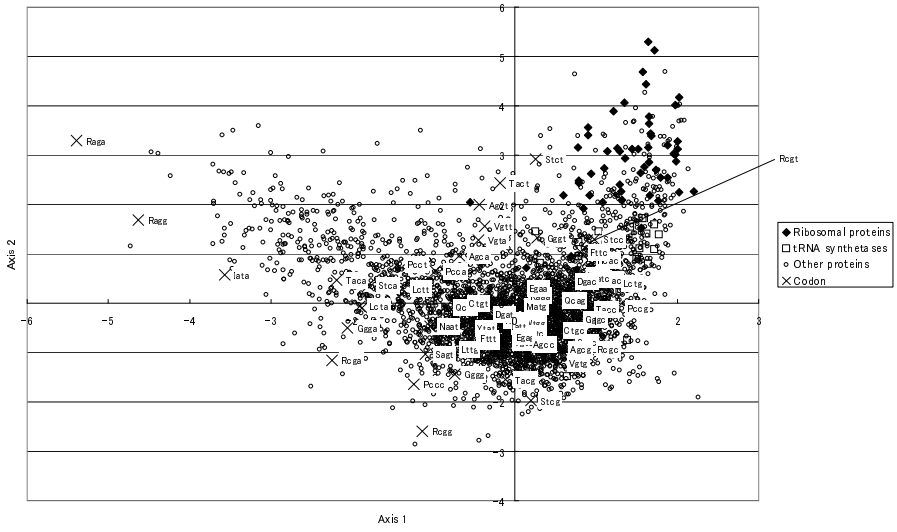


図 6.10　大腸菌のコドン使用の対応分析結果

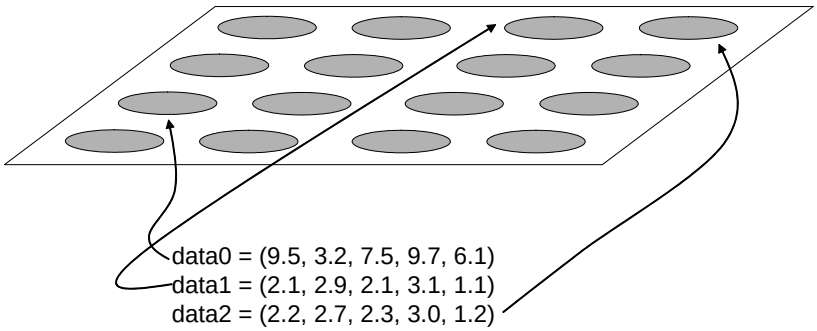


図 6.11　各データの二次元平面上へのマッピング

が分かります。一方で aga,agg は左の方に位置しています。この結果より、リボソームタンパク質や tRNA シンセターゼの分布を右にシフトさせている要因として cgc や cgt の使用率が考えられます。実際にリボソームタンパク質や tRNA シンセターゼのコドン使用を見てみると、アルギニンに対応するコドンとして cgc や cgt が多く使われ、逆に aga や agg はほとんど使われていません。

このように対応分析を使うとデータの次元を落とすだけでなく、行と列の関係を同じ座標上で把握することができます。ただ対応分析は本来クロス集計表に対して行われる手法であり、RSCU 値などに対して行う場合はその解釈に十分な注意が必要です [30]。

6.7　自己組織化

自己組織化 (SOM, Self Organization Map) は与えられた多次元のデータ群の 1 つ 1 つを二次元平面など低い次元に対応される手法です [32], [33]。平面上では類似したデータ同士が近くに配置されるようにします。このため、データ間の類似関係が感覚的に大変分かりやすくなります。クラスターというものは特に定義されないため、k 平均アルゴリズムと異なり、あらかじめクラスター数を指定する必要がありません。逆に言えば、どのデータがどのクラスターに属するかということは、通常は不明瞭です。

図 6.11 に自己組織化の基本的な考え方を示します。5 次元データが 3 つ与えられており、data1 と data2 が近い値を示しています。自己組織化は各データを図の上に示す二次元平面上のどこかにマッピングしますが、このとき data1 と data2 をなるべく近づけるようにマッピングします。以下に各データを効果的にマッピングする方法について述べます。

自己組織化では図 6.12 に示すように、まず出力層と呼ばれる二次元平面上のノード、入力層と呼ばれるデータ入力用のノード、そして全ての出力層のノード

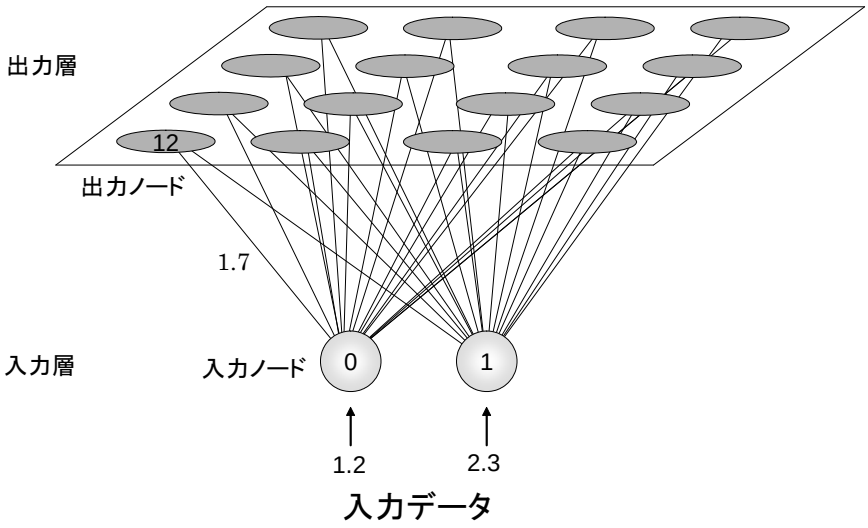


図 6.12　自己組織化マップの構造

と全ての入力層のノードをつなぐ結線を定義します。上の層は各データを対応させる平面とし、下の層は各データを直接受け取ります。ここで入力層のノードの数をデータの次元数 n と等しくします。そして入力 i と出力層のノード j の間の結線に重み w_{ij} を割り当てます。次に n 次元のデータ群を時刻 $t\,(=0,1,2,\cdots)$ において１つずつネットワークに提示してゆきます。そして提示されたデータに基づいて w_{ij} を更新していきます。同時に各ノード j の近傍として定義される領域 N_j の範囲も狭めてゆきます。詳細なアルゴリズムは以下の通りです。

$w_{ij}(t)\,(0 \le i \le n-1)$ を、時刻 t における入力 i からノード j への結合の重みとします。また $N_j(t)$ を時刻 t におけるノード j の近傍とします。入力データを x とし、その i 次元目 $(0 \le i \le n-1)$ の数値を x_i とします。

1. 初期化 w_{ij} を乱数を使って初期化し、$N_j(0)$ を大きく設定する。

2. 入力データの提示 時刻 t におけるノード j への入力データ $x(t)$ を入力層に提示する (図 6.13(a))。

3. 距離の計算 以下の式により、入力データと出力ノード j との距離 d_j を計算する。

$$d_j = \sum_{i=0}^{n-1} \left(x_i(t) - w_{ij}(t)\right)^2$$

例えば、図 6.13(a) の入力データと、一番左の出力ノードとの距離は、$d_0 = (x_0 - w_{0\ 0})^2 + (x_1 - w_{1\ 0})^2 = (1.2-0.7)^2 + (2.3-1.5)^2 = 0.89$ である。これを全ての出力ノードに対して計算する。

4. 最小距離の選択 d_j が最小となる出力ノード $j_{\min}$ を特定する (図 6.13(b))。

5. 結合荷重の更新 ノード $j_{\min}$ の近傍ノードの集合 $N_{j_{\min}}(t)$ を決定する (図 6.13(c))。そして、ノード $j_{\min}$ と $N_{j_{\min}}(t)$ への結合荷重を以下の式により更新する。

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$

ここで、j は $N_{j_{\min}}(t)$ に含まれる全てのノードである。また $\eta(t)$ はゲイン $(0 < \eta(t) < 1)$ と呼ばれる。例えば、$\eta(t) = 0.5$ のとき、図 6.13(d) のように入力ノード 0,1 から $j_{\min}$ への重みは、$w_{0\ j\min} = 1.1 + 0.5(1.2-1.1) = 1.15$, $w_{1\ j\min} = 1.9 + 0.5(2.3-1.9) = 2.1$ で、それぞれ 1.15 と 2.1 になる。

6. 反復 t を増加させて入力データを変え、2.〜5. を繰り返す。このとき、$(0 < \eta(t) < 1)$ を時間とともに減少させ、重みの更新速度を減少させる。近傍 $N_{j^{\min}}(t)$ も、時間とともにサイズを減少させ、更新される領域を狭めていく。

以上をごく簡単に要約すると以下のようになります。

- 学習データに最も近い重みベクトルを持つユニットを探す。

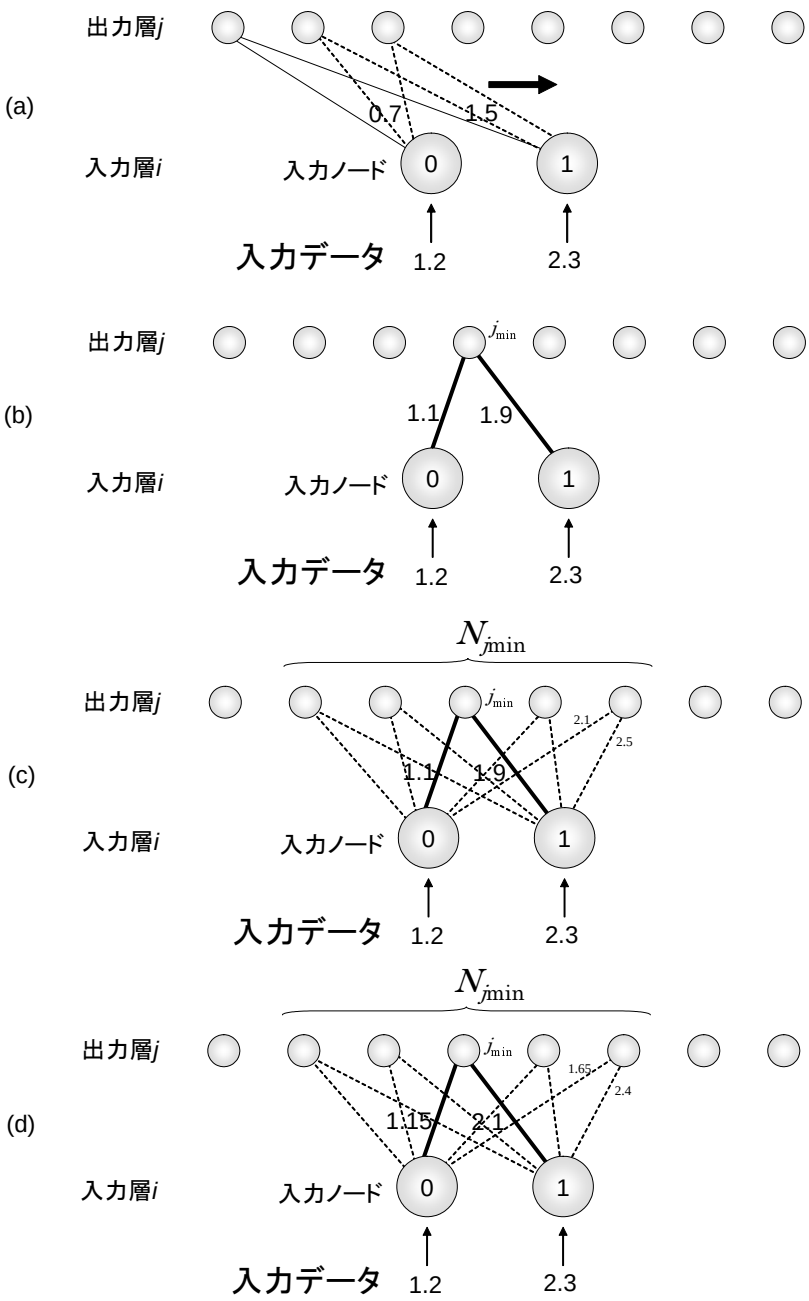


図 6.13　自己組織化アルゴリズム

- このユニットの重みベクトルと入力データの近似度を増加させ、その近傍のユニットについても、重みと入力との距離を減少させる。

最終的に各データは「4. 最小距離の選択」に基づき、d_j が最小となる出力ノードにマッピングされます。

自己組織化を実装すると例えば以下のようなプログラムになります。

```
/*
(1) X_LEN, Y_LEN はそれぞれマップの横および縦の長さ。
(2) INPUT_DIM は入力データの次元数。
(3) MAX_TIME は反復回数。
ともに#define で定義すること。
*/
void self_org(double **data_set, int data_num){

  static double w[X_LEN * Y_LEN][INPUT_DIM];
     /* w[j][i], 入力層のノード i から出力層 j への重み */
  static double inp[INPUT_DIM];
     /* 入力データ格納用の配列変数 */

  int t; /* 時刻 t */
  int i, j, min_j;
  int k;
  double dist, min;

  static int neighbours[X_LEN * Y_LEN];
     /* 近傍ノード格納用の配列変数 */

  int n_neighbours; /* 近傍ノードの数 */

  /* w[][] の初期化 */
  for(j = 0;j < X_LEN * Y_LEN;j ++)
    for(i = 0;i < INPUT_DIM;i ++)
      w[j][i] = 1.0 * (rand() % 100);


  for(t = 0;t < MAX_TIME;t ++){
    next_data(inp, t, data_set, INPUT_DIM, data_num);
       /* 時刻 t における次元 INPUT_DIM の入力データを inp に入れる */
```

#	d1	d2	d3
0	4.0	5.0	2.0
1	4.1	5.2	2.3
2	70.0	70.0	70.0
3	4.3	5.0	2.1
4	20.0	21.0	22.0
5	3.8	5.3	2.0
6	1.0	2.0	3.0
7	5.0	4.0	3.0
8	1.0	3.0	3.0
9	1.0	2.1	3.1
10	5.1	4.1	3.1
11	1.1	3.1	3.1
12	50.0	34.0	13.0
13	45.0	32.0	15.0
14	49.0	51.0	21.0
15	3.9	5.0	2.1
16	21.0	22.0	23.0
17	23.0	24.0	25.0
18	25.0	26.0	27.0
19	4.5	5.1	2.2

(a)

3		19		7		10			2
0,15		5		1					
11				16				14	
	8								
6					17				
9		4						13	
						18			12

(b)

図 6.14　自己組織化の実行結果の例

20 個の三次元データを自己組織化にかけた結果。(a) 自己組織化にかけたデータ。#はデータ番号。d1, d2, d3 はそれぞれの次元の値。(b) 自己組織化の結果。数字はデータ番号を表す。マスの中の数字で、0,15 は 0 番目と 15 番目のデータが同じところにマップされたことを表す。

```
    min_j = find_winner(inp, w);
       /* 勝者ユニットの決定 */

    n_neighbours = neighbour(min_j, t, neighbours);
       /* 近傍ノードを neighbours に、その数を n_neighbours に入れ
る */

    /* 勝者ノードを含む近傍ノードに関する重みの更新 */
    for(k = 0;k < n_neighbours;k ++){
      j = neighbours[k];
      for(i = 0;i < INPUT_DIM;i ++){
         w[j][i] += update_rate(t) * (inp[i] - w[j][i]);
      }
    }
   }
}
```

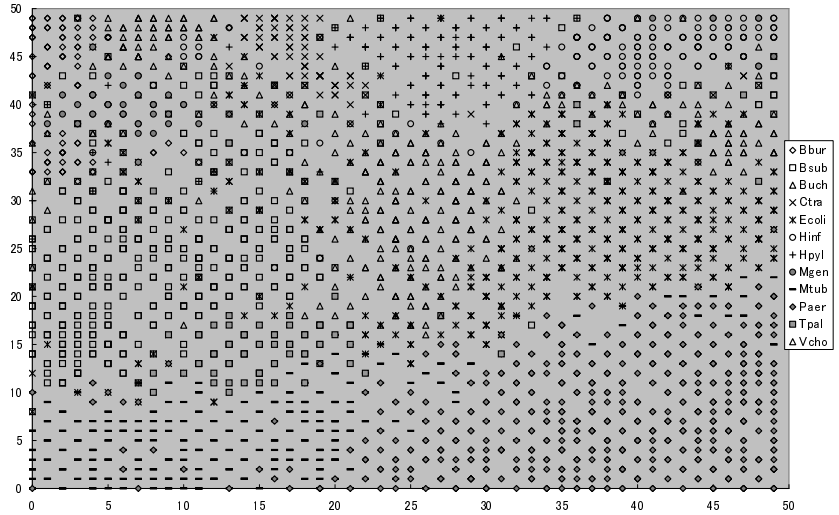


図 6.15　コドンバイアスの自己組織化

12 種類のバクテリアの遺伝子のコドンバイアスを自己組織化にかけた結果。全遺伝子の中から、1/10 の遺伝子を無作為に抽出したものを使用した。解析した生物種は以下の通り。Bbur: *Borrelia burgdorferi*, Bsub: *Bacillus subtilis*, Buch: *Buchnera aphidicola*, Ctra: *Chlamydia trachomatis*, Ecoli: *Escherichia coli*, Hinf: *Haemophilus influenzae*, Hpyl: *Helicobacter pylori*, Mgen: *Mycoplasma genitalium*, Mtub: *Mycobacterium tuberculosis*, Paer: *Pseudomonas aeruginosa*, Tpal: *Treponema pallidum*, Vcho: *Vibrio cholerae*。

図 6.14 に 20 個の 3 次元データを SOM によって 2 次元平面上にマップした結果を示します。これを見ると例えば、6,8,9,11 は近い値を持っており、これらがマップ上で比較的近い位置に配置されていることが分かります。

6.8　コドンバイアスの自己組織化

では自己組織化を使って、似たようなコドンバイアスを持つ遺伝子を集めてみましょう。ここでは大腸菌や枯草菌、マイコプラズマ菌など 12 種類の生物の各遺伝子の 61 次元の RSCU 値を求め、それらを一緒に自己組織化にかけました。その結果を図 6.15 に示します。

各点は各遺伝子を表し、似たコドンバイアスを持つもの同士が集まるように配置されているはずです。コドンバイアスは概ね種ごとに集まっていることが分かります。また大腸菌 (Ecoli, *E. coli*) とコレラ菌 (Vcho, . *V. cholerae*) は

コドンバイアスがやや似ていますが、枯草菌 (Bsub, *B. subtilis*) とは離れているなど、種間のコドンバイアスの類似性なども読み取れます。大腸菌の遺伝子でも時折離れたところに配置されているものがありますが、これは異なるコドンバイアスを持つ外来遺伝子の可能性が考えられます。

このように自己組織化を使うことによって、各遺伝子のコドンバイアスの特徴を二次元平面上にマッピングして分かりやすく比較することが可能になっています。

演習問題

6.1 6.6 のプログラムをベースにして、コドンバイアスなど多次元データの主成分分析を行ってみましょう。

6.2 6.9 のプログラムをベースにして、コドンバイアスなど多次元データの対応分析を行ってみましょう。

6.3 行列 F の任意の列を n 倍しても、主成分分析の結果は変わらないことを示しましょう。

6.4 自己組織化を行うプログラムを完成させましょう。

付録 A
再帰アルゴリズムの基本

再帰アルゴリズムを使うと解く問題によっては、アルゴリズムを簡潔に表現をすることができます。本テキストで扱っているアラインメント、RNA 二次構造予測、隠れマルコフモデルで使われるアルゴリズムは、再帰アルゴリズムを用いて簡潔に記述することができます。そこでこの章で再帰アルゴリズムを十分にマスターしておきましょう。

A.1 階乗計算の再帰的定義

再帰は関数の中で自分自身の関数を呼び出すことを意味します。例えば、関数 f の定義の中で f を呼び出していれば、f は再帰的な関数ということができます。これをまず、階乗の計算を例にとって説明しましょう。

階乗というのは 1 からある数までの積をいいます。例えば、

- 9 の階乗は $9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$
- 10 の階乗は $10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$、
- n の階乗は $n \times (n-1) \times (n-2) \cdots\cdots \times 1$

となります。n の階乗は $n!$ と表します。

さて $n!$ を厳密に定義するには、どうしたらいいでしょうか？一番分かりやすい定義は、

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

と表すことでしょう。でも $\cdots$ の部分が直感的であまり厳密な意味を持たない気がしませんか。しかし数学的には以下のように厳密に定義することができます。

$$n! = \begin{cases} 1 & (n=1) \quad \cdots \quad (1) \\ n \times (n-1)! & (n>1) \quad \cdots \quad (2) \end{cases}$$

こうすると例えば、3!は、

1. (2) より、3! = 3 x 2!
2. (2) より、3 x 2! = 3 x 2 x 1!
3. (1) より、3 x 2 x 1! = 3 x 2 x 1

となります。

再帰を使ったプログラムを 4 の階乗の計算を例として書くと、以下のようになります。

```
#include<stdio.h>

int factorial(int n){
     if(n==1)n=1;                  /* 式 (1)  */
     else n=n*factorial(n-1);      /* 式 (2)  */
     return n;
}
main(){
     int a;
     a=factorial(4);
     printf(" %d\n ",a);
}
```

factorial という関数の中で、さらに factorial という関数を呼び出しているのが分かると思います。再帰の様子を表したのが図 A.1 です。4 回 factorial 関数が呼び出され、引数 n に値が代入されています。n は**ローカル変数**と呼ばれるものです。これはその関数が呼ばれた段階で局所的に確保される変数です。噛み砕いて言えば、呼び出す度に異なる変数になります。従って再帰 1、2、3 のそれぞれにおける変数 n は別のものなので、例えば再帰 2 の段階で n に 2 が代入されても、再帰 1 で定義された n の値は変化しません。

再帰プログラムの特徴は２つあります。１つは、関数の中で自分の関数を呼び出していること、もう１つは、関数の中に自分の呼び出しをストップされるような選択肢があることです。階乗計算の例で言えば、(1) が前者、(2) が後者となります。後者がなければ、関数は再帰を繰り返して永遠に回ってしまうことになります。

A.2　フィボナッチ数列計算の再帰的定義

再帰的アルゴリズムの次の例として、フィボナッチ数列について解説します。フィボナッチ数列とは、ピサのレオナルドの異名を持つ数学者、レオナルド・

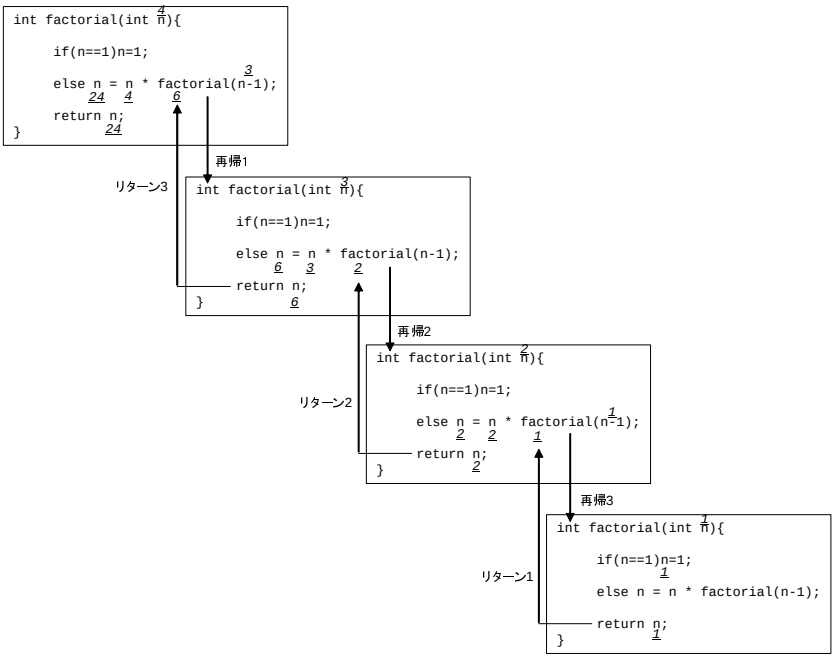


図 A.1　factorial の再帰の様子

関数 factorial(4) を呼び出したときの再帰の様子を表し、再帰１→再帰２→再帰３→リターン１→リターン２→リターン３の順にプログラムが流れる。変数または式の値を下に示した（引数に関わる数値は上に示した）。

フィボナッチによって発見された数列です。フィボナッチはその著「算術の書」のなかで次のような話をしました。

1. 生まれたばかりの１つがいのウサギがいます。
2. この１つがいのウサギは、２か月目から毎月１つがいのウサギを生むとします。
3. 生まれたウサギも２か月すると、同じように毎月１つがいの子を生むとします。

このようにして増え続けていくと、毎月のウサギのつがいの数はどうなっていくでしょうか。これを数列として表すと、1,1,2,3,5,8,13,... となる数字をたどっていくことになります。つまり 2 つ前の数値と 1 つ前の数値を足すと、現在の数値になるのです。フィボナッチ数列の n 番目の数値 fibonatcci(n) を数式で表現すると、以下のようになります。

$$\text{fibonatcci}(n) = \begin{cases} 1 & (n=1) \quad \cdots \quad (1) \\ 1 & (n=2) \quad \cdots \quad (2) \\ \text{fibonatcci}(n-1) & \cdots \\ \qquad +\text{fibonatcci}(n-2) & (n>2) \quad \cdots \quad (3) \end{cases} \tag{A.1}$$

これをそのまま関数として実装すると、以下のようになります。

```
#include <stdio.h>

int fibonatcci(int n){
  int ret;
  if(n == 1)ret = 1; /* 式 (1) */
  else if(n == 2)ret = 1; /* 式 (2) */
  else ret = fibonatcci(n - 1) + fibonatcci(n - 2); /* 式 (3) */
  return ret;
}

main(){
  printf("%d\n", fibonatcci(8));
}
```

A.3 計算の重複の回避

fibonatcci 関数をそのまま実装しただけでは、求める数が大きくなった場合、プログラムの実行速度が非常に遅くなります。そこでプログラムを高速化する

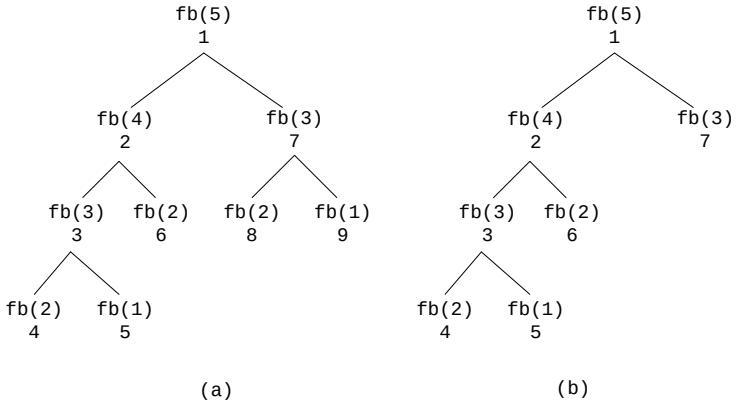


図 A.2　関数 fibonatcci の再帰

関数 fibonatcci(fb と略記) の引数に代入される値が括弧の中に、呼び出される順番が関数の下に書かれている。(a) は途中計算の記録をしない場合の関数の呼び出され方で、(b) が途中計算の記録をする場合。(b) の方では fb(3) で一度再帰が起こると次からは再帰が起こらなくなり、結果として関数 fibonatcci の呼び出し回数が減っていることが分かる。

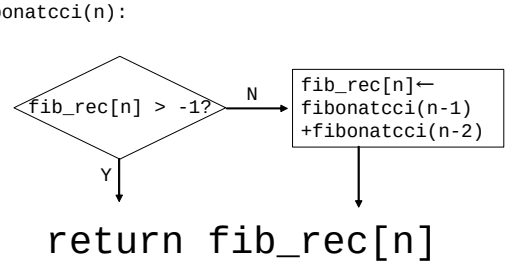


図 A.3　関数 fibonatcci の計算過程の記録アルゴリズム

fibonatcci(n) が呼ばれたとき、`fib_rec[n]` の値をまず見る。そこに計算済みの答えがある場合 (`fib_rec[n]` > -1)、`fib_rec[n]` をそのまま返す。そうでなければ、再帰計算を行い、その答えを `fib_rec[n]` に記録した上で返す。n が 1 または 2 のときの動作は省略した。

方法を考えましょう。図 A.2(a) は fibonatcci(5) が呼ばれたとき、どのように再帰が起こっているかを示しています。よく見てみると、fibonatcci(3) が 3 回目と 7 回目に呼び出されているのが分かります。fibonatcci(3) の値は 3 回目の呼び出し時に一度求めていますから、7 回目の呼び出し時にもう一度再帰して答えを求めるのは効率が悪いと言えます。そこで fibonatcci(3) の値を保存しておけば、次回 fibonatcci(3) が呼ばれたときにもう一度再帰することなく、保存してある値を答えとして返すことができます。結果として無駄な再帰が起こらなくなるので、プログラムの高速化が実現できます (図 A.2(b))。

これをプログラムとして実装するには、途中計算結果を保存しておく配列変数 `fib_rec[]` を定義し、初期値-1(負の値なら何でもよい) を代入しておきます。そして fibonatcci(n) の値が計算されたときに、`fib_rec[n]` にその値を代入します。そして fibonatcci(n) が呼ばれる度に `fib_rec[n]` の値を確認し、初期値-1 でなければ、`fib_rec[n]` には fibonatcci(n) の値が入っていることになるので、`fib_rec[n]` の値をそのまま返せばいいことになります。図 A.3 にそのアルゴリズムの概要を示しました。以下が実装例です。

```
#include <stdio.h>
#define MAX_REC 1000000

static int fib_rec[MAX_REC];
   /* fibonatcci の計算結果を入れておく配列変数 */

int fibonatcci(int n){
```

```
   int ret;

   if(fib_rec[n] > 0)return fib_rec[n];
      /* fib_rec[n] が-1 でなければ、fib_rec[n] に fibonatcci(n) の
         答えが入っている */

   if(n == 1)ret = 1; /* 式 (1) */
   else if(n == 2)ret = 1; /* 式 (2) */
   else ret = fibonatcci(n - 1) + fibonatcci(n - 2); /* 式 (3) */

   fib_rec[n] = ret; /* 再帰による計算結果を fib_rec[n] に保存 */
                     /* 次回からはこの中に入っている値を返す */
   return ret;
}

main(){
   int i;
   for(i = 0;i < MAX_REC;i ++)
      fib_rec[i] = -1; /* 配列変数 fic_rec[] の初期化
                          (-1 を入れておく) */
   printf("%\n", fibonatcci(8));
}
```

上記プログラムを実装して演習問題 1.1,1.2 を解いてみましょう。

式 A.1 をそのまま実装すると再帰関数をなってしまいますが、再帰を使わない方法もあります。fibonatcci(n) の値は fibonatcci(n-1) と fibonatcci(n-2) の値が分かっていれば求めることができます。逆に言えば、fibonatcci(n-1) と fibonatcci(n-2) の値を**先に**求めてしまえば、fibonatcci(n) の値は直ちに求めることができます。

再帰計算では、fibonatcci(n) の値を要求し、その次に fibonatcci(n-1) と fibonatcci(n-2) の値を要求してこの 2 つの値が求まったところで、fibonatcci(n) の値を求めていました。この要求するというステップ (関数の再帰呼び出し) を省き、fibonatcci(0)=0, fibonatcci(1)=1 として、計算を fibonatcci(2), fibonatcci(3), $\cdots$ と「下から積み上げていく」ことにより、再帰を使わずかつ効率の良いプログラムを作成することができます。以下がその例です。

```
int fibonatcci(int n){
  int i;
```

```
  fib_rec[0] = 0;
  fib_rec[1] = 1;
  for(i = 2;i <= n; i ++)
    fib_rec[i] = fib_rec[i - 1] + fib_rec[i - 2];
  return fib_rec[n];
}
```

A.4 計算のオーダー

n を引数としたとき、fibonatcci という関数が呼ばれる回数 F_r は途中計算結果を記録しないとき、

$$F_r(n) = 1 - \frac{1}{\sqrt{5}}\left\{(3+\sqrt{5})\left[1-\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}\right] - (3-\sqrt{5})\left[1-\left(\frac{1-\sqrt{5}}{2}\right)^{n-2}\right]\right\}$$

になります[*1)]。ところが、途中計算結果を記録すると、回数を $2n-3$ $(n \geq 2)$ まで減らすことができます。ここでオーダーの考え方を使ってこの計算量をもう少し分かりやすく表現しましょう。

ある定数 c が存在して十分大きな n に対し、$g(n) < cf(n)$ が成立するとき、「$g(n)$ は $O(f(n))$」と書き、$g(n)$ は $f(n)$ のオーダーであると呼びます。先ほどの例で $g(n) = 2n-3$ とおくと、$c = 3$、$f(n) = n$ とすれば、$n > -3$ に対し $g(n) < cf(n)$ が成立するので、途中計算を記録する場合の計算量は $O(n)$ となります。同様にして、定数 c を十分大きく設定すれば、十分大きな n に対して $c \cdot 2^n > F_r(n)$ が成立するため、途中計算を記録しない場合の計算量は $O(2^n)$ となります。明らかに途中計算を記録した方が計算量が少なくなるのが分かると思います。

A.5 ハノイの塔

階乗やフィボナッチ数列の計算例では、再帰のメリットが少し分かりづらかったと思います。しかし再帰はときとしてプログラムの簡潔な記述を可能にします。それを「ハノイの塔」を題材にしてみてゆきましょう。

ハノイの塔は、今から 100 年くらい前に、フランスのリュカという人が考え出したゲームです。台の上に柱が 3 本立っていて、そのうちの 1 本に円盤が何枚か、大きい順にはまっています。この円盤をルールに従って、そっくり別の柱に移すゲームです。

---

*1) $a_1 = 1, a_2 = 1, a_n = a_{n-1} + a_{n-2} + 1, n > 2$ として、a_n の一般項を求めるとこのような数式になる。

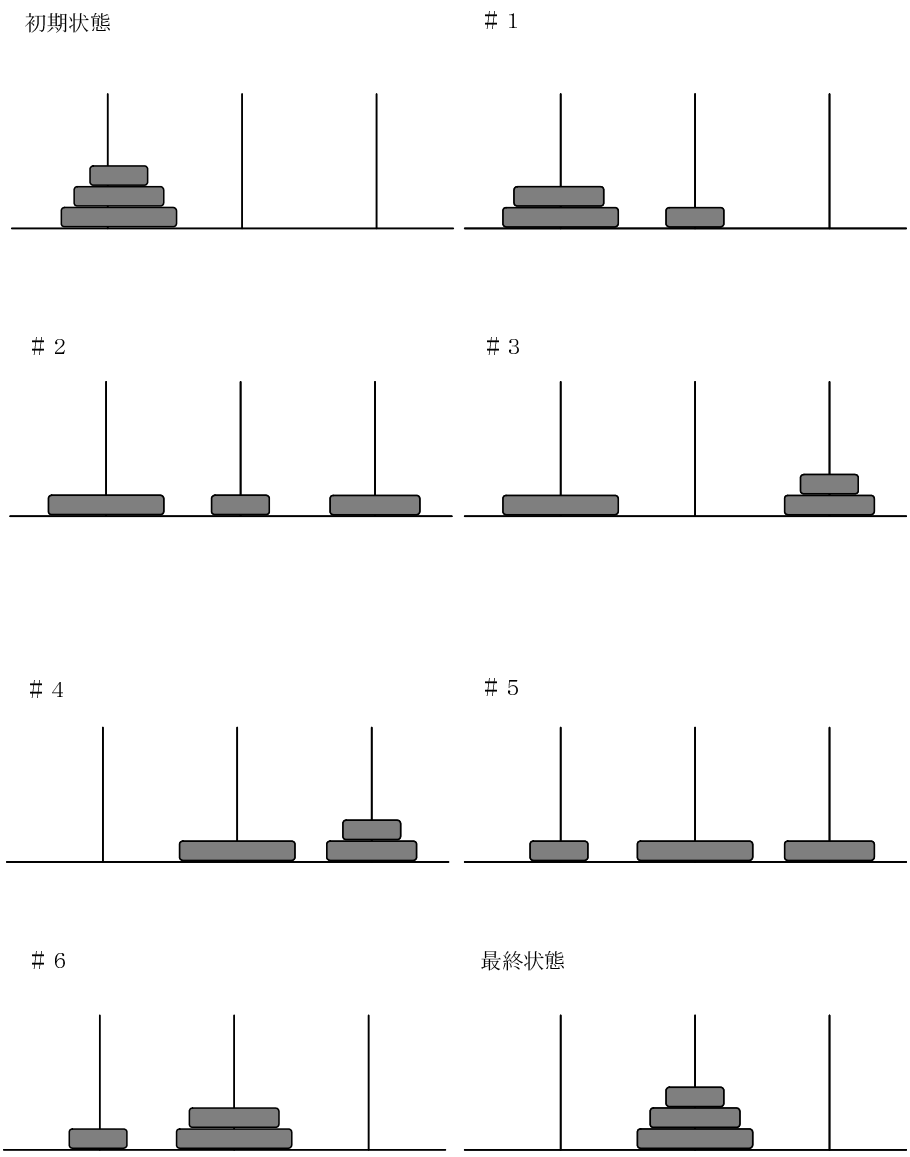


図 A.4　円盤が３枚のときの移し方

円盤の移し方のルールは以下の通りです。

1. １回に１枚の円盤しか動かしてはいけません。
2. 小さい円盤の上に大きい円盤を載せてはいけません。つまり、円盤を置くときは、必ずその円盤より大きな円盤の上に置かなければなりません。
3. 円盤はどれかの柱にかならずはめなければなりません。手に持っていたり、下においたりしてはいけません。

こうしてできるだけ少ない回数で別の柱へ円盤を全部移すようにします。

図 A.4 は円盤が３枚のときの移し方です。

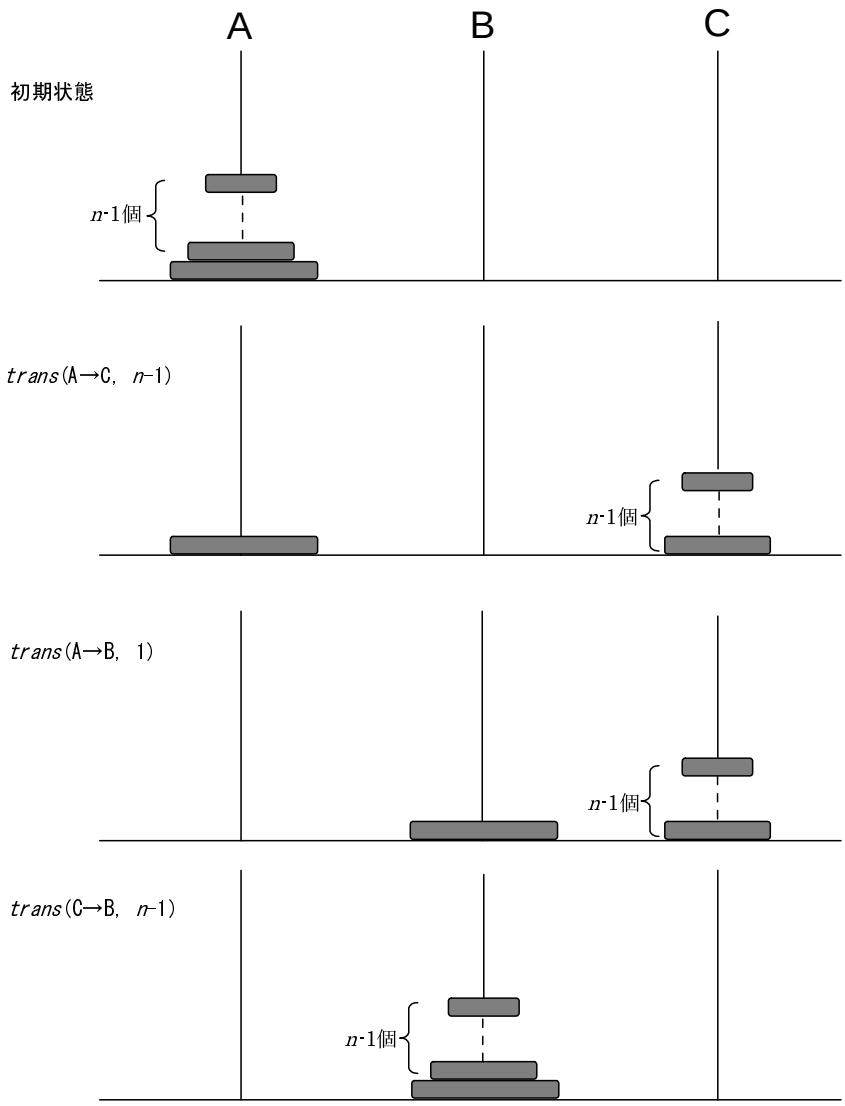


図 A.5　円盤が n 枚のときの移し方

ここでどの棒からどの棒へどのような順序で円盤を移せばいいかを求めるアルゴリズムを考えましょう。今棒 α、β、γ があるとして[*2)]、α に置かれている円盤 n 個を β に移す操作を $trans(\alpha \rightarrow \beta, n)$ と表します。

棒 α=A に置かれた n 個の円盤を γ=C を中継して棒 β=B に移すときには、図 A.5 に示すように、まず A の上の $n-1$ 個の円盤を一回 C に移し、A の一番下の円盤を B に移してから、C に退避させた $n-1$ 個の円盤を B に移します。この操作は再帰を使って以下のように非常に簡潔に表すことができます。

*2)　α、β、γ は棒を表す変数として使っている。A,B,C は棒を表す定数として使っている。

$$trans(\alpha \to \beta, n) = trans(\alpha \to \gamma, n-1) + trans(\alpha \to \beta, 1) + trans(\gamma \to \beta, n-1)$$

但し操作 1+操作 2 は、操作 1 を行った後に操作 2 を行うことを意味します。ちなみに $trans(\alpha \to \beta, 1)$ は、棒 α の一番上の円盤を一枚取って、棒 β に移すことを意味します。プログラムとして実装するときは、円盤が移ったことを表示したり (演習問題 1.3)、メモリ上で円盤の位置を移し変えたり (演習問題 1.4) する動作に該当します。演習問題を通じてハノイの塔の円盤を移し変える手順を求めるアルゴリズムをぜひ実装してみて下さい。

演習問題

1.1 フィボナッチ数列の 36 項目の数字（36ヶ月目のつがいの数）はいくつになるでしょうか？フィボナッチ数列を計算するプログラムを使って調べましょう。

1.2 フィボナッチ数列が 10,000,000 を超えるのは、第何項目でしょうか？

1.3 ハノイの塔で、n 枚の円盤を柱 from から柱 to へ移すときの操作を表示する再帰的関数 trans(char from, char to, int n) を作りましょう。trans('A', 'B', 3) の出力結果は以下のようになるはずです。

```
A->B
A->C
B->C
A->B
C->A
C->B
A->B
```

1.4 ハノイの塔で棒 A に下から順（大きい順）に円盤 a,b,c が刺さっているとします。これを柱 B に移し変えるときの状態の移り変わりを表示するプログラムを作りましょう。以下のような出力になるはずです。

```
A:a,b,c
B:
C:

A:a,b
B:c
C:

A:a
```

```
B:c
C:b

A:a
B:
C:b,c

A:
B:a
C:b,c

A:c
B:a
C:b

A:c
B:a,b
C:

A:
B:a,b,c
C:
```

1.5　ハノイの塔で、移す円盤の数を n としたときにかかる計算量のオーダは？

付録 B
統計解析に関する補足

B.1 標準正規分布

n 本の塩基配列の中に、ある配列パターンが有意に出現するかを統計的に検証することを考えます。今 p を与えられた一本の配列であるパターンが (偶然に) 観測される確率とします。n 本中、ちょうど x 本の配列にそのパターンが含まれる確率は二項分布を使って、

$$P(x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{B.1}$$

と表されます。n が大きくなると、$P(x)$ はどのような分布になっていくのでしょうか。これを以下で説明してゆきます [34], [35]。
まず x を以下の式で z に正規化します。

$$z = \frac{(x - np)}{\sqrt{np(1-p)}} \tag{B.2}$$

このとき z が従う分布を $g(z)$ とし、$g(z)$ を求めてゆきましょう。x は $0, 1, 2, \cdots$ と離散的な値をとるので、x の変化量 Δx の最小値は 1 です。このとき、z の変化量 Δz は常に

$$\Delta z = \frac{1}{\sqrt{np(1-p)}} \tag{B.3}$$

となります。式 B.2 を式 B.3 で割って変形すると、

$$x = np + \frac{z}{\Delta z} \tag{B.4}$$

が得られます。

式 B.1 の x に $x+1$ を代入すると、

$$P(x+1) = \binom{n}{x+1} p^{x+1} (1-p)^{n-x-1} \tag{B.5}$$

ここで、

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

を使って式 B.5 を式 B.1 で割り、式 B.6 を代入すると、

$$\frac{P(x+1)}{P(x)} = \frac{(n-x)p}{(x+1)(1-p)} \tag{B.6}$$

となります。さてここで $P(x)$ と $g(z)$ の関係を考えましょう。z と x は式 B.2 のような関係で結ばれています。さて一般的に離散的な値を取る確率変数 X が確率密度関数 P_d に従うとき、$X = x$ である確率は $P_d(x)\Delta x$ で求められます。但し、Δx は確率変数 X が取り得る値の中で $> x$ を満たす最小のものと x との差です[*1)]。従って $g(z)$ が確率密度関数であるためには、$g(z)\Delta z$ が確率 $P(x)\Delta x = P(x)$ を表さなければなりません。すなわち、

$$g(z) = \frac{1}{\Delta z}P(x) \tag{B.7}$$

および

$$g(z+\Delta z) = \frac{1}{\Delta z}P(x+1) \tag{B.8}$$

が成立します。式 B.8 を式 B.7 で割り、式 B.6 を代入すると、

$$\frac{g(z+\Delta z)}{g(z)} = \frac{P(x+1)}{P(x)} = \frac{(n-x)p}{(x+1)(1-p)} \tag{B.9}$$

式 B.9 に B.4 を代入して x を消去すると、

$$\frac{g(z+\Delta z)}{g(z)} = \frac{(n-np-z/\Delta z)p}{(np+z/\Delta z+1)(1-p)} = \frac{np(1-p)-pz/\Delta z}{np(1-p)+(1-p)(z/\Delta z+1)} \tag{B.10}$$

式 B.3 より $np(1-p) = \frac{1}{\Delta z^2}$ なので、これを上の式に代入して、

$$\frac{g(z+\Delta z)}{g(z)} = \frac{\frac{1}{\Delta z^2} - p\frac{z}{\Delta z}}{\frac{1}{\Delta z^2} + (1-p)(\frac{z}{\Delta z}+1)} = \frac{1-pz\Delta z}{1+(1-p)\{z\Delta z+\Delta z^2\}} \tag{B.11}$$

となります。式 B.11 を使うと、

$$\begin{aligned}
\frac{g(z+\Delta z)-g(z)}{\Delta z} &= \left(\frac{g(z+\Delta z)}{g(z)} - 1\right)\frac{g(z)}{\Delta z} \\
&= \frac{1-pz\Delta z-1-(1-p)(z\Delta z+\Delta z^2)}{1+(1-p)(z\Delta z+\Delta z^2)}\frac{g(z)}{\Delta z} \\
&= \frac{-z\Delta z-(1-p)\Delta z^2}{1+(1-p)(z\Delta z+\Delta z^2)}\frac{g(z)}{\Delta z} = \frac{-z-(1-p)\Delta z}{1+(1-p)(z\Delta z+\Delta z^2)}g(z)
\end{aligned} \tag{B.12}$$

が得られます。式 B.3 より $n \to \infty$ のとき、$\Delta z \to 0$ となります。このとき、式 B.12 の極限は

$$\frac{dg(z)}{dz} = -zg(z) \tag{B.13}$$

---

*1) $x \le X < x + \Delta x$ において、$P_d(X)$ は一定と考える。

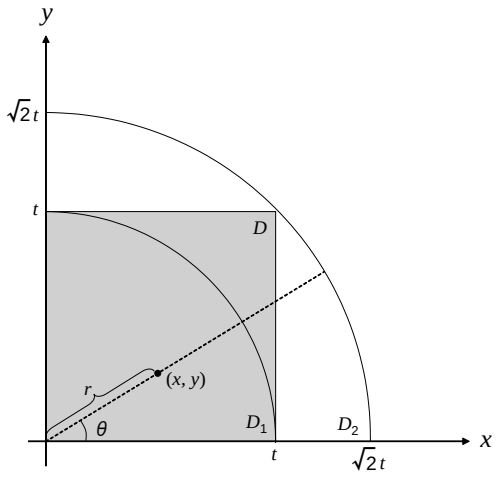


図 B.1　積分を行う領域

D が灰色の正方形の領域。D_1 は内側の扇形、D_2 は外側の扇形である。D_1の面積 $<$ D の面積 $<$ D_2の面積 となる。

となります。これを解くと、

$$g(z) = Ce^{-\frac{1}{2}z^2}$$

が得られます。但し C は定数です。以下、この C を定めましょう。$g(z)$ は確率密度関数なので、

$$\int_{-\infty}^{+\infty} g(z)dz = \int_{-\infty}^{+\infty} Ce^{-\frac{1}{2}z^2}dz = 1 \tag{B.14}$$

が成立します。$I(t) = \int_0^r e^{-\frac{1}{2}z^2}dz$ とおくと、

$$I(t)^2 = \left\{\int_0^t e^{-\frac{1}{2}z^2}dz\right\}^2 = \int_0^t \int_0^t e^{-\frac{1}{2}(x^2+y^2)}dxdy$$

ここで、積分を行う領域を図 B.1 のように考えると、

$$I(t)^2 = \int\int_D e^{-\frac{1}{2}(x^2+y^2)}dxdy$$

となります[6]。

$$I_1(t)^2 = \int\int_{D_1} e^{-\frac{1}{2}(x^2+y^2)}dxdy, \quad I_2(t)^2 = \int\int_{D_2} e^{-\frac{1}{2}(x^2+y^2)}dxdy$$

とすれば、

$$I_1(t)^2 < I(t)^2 < I_2(t)^2$$

です。さて図 B.1 のように、

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$$

のような変数変換ができるので、

$$I_1(t)^2 = \int\int_{D_1} e^{-\frac{1}{2}r^2} \left| \begin{array}{cc} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} \end{array} \right| drd\theta = \int_0^{\frac{\pi}{2}} \int_0^t e^{-\frac{1}{2}r^2} r dr d\theta$$

$$= \left[\left[-e^{-\frac{1}{2}r^2}\theta \right]_{r=0}^{r=t} \right]_{\theta=0}^{\theta=\frac{\pi}{2}} = \frac{\pi}{2} - \frac{\pi}{2}e^{-\frac{1}{2}t^2}$$

同様にして、

$$I_2(t)^2 = \int_0^{\frac{\pi}{2}} \int_0^{\sqrt{2}t} e^{-\frac{1}{2}r^2} r dr d\theta = \frac{\pi}{2} - \frac{\pi}{2}e^{-t^2}$$

従って、$\lim_{t\to\infty} I_1(t)^2 = I_2(t)^2 = \frac{\pi}{2}$ となるので、$\lim_{t\to\infty} I(t)^2 = \frac{\pi}{2}$、$\lim_{t\to\infty} I(t) = \sqrt{\frac{\pi}{2}}$ となります。よって、

$$\int_{-\infty}^{+\infty} e^{-\frac{1}{2}z^2} dz = \sqrt{2\pi}$$

となり、$C = \frac{1}{\sqrt{2\pi}}$ が導かれます。結局 z の確率密度関数は、

$$g(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$$

になることが導かれました。この確率密度関数に従う分布を標準正規分布と呼びます。

さて、正規分布 $N(\mu, \sigma^2)$ の確率密度関数は、

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{B.15}$$

でなので、

$$z = \frac{x - np}{\sqrt{np(1-p)}} \quad \sim \quad N(0,1) \tag{B.16}$$

となります。

B.2 カイ自乗分布

確率変数 X が標準正規分布に従うとき、$Y = X^2$ はどのような分布に従うのでしょうか？Y の分布関数 $G(y) = P(Y \leq y)$ を用いると、$y \geq 0$ に対して、

$$G(y) = P(Y \leq y) = P(X^2 \leq y) = P(-\sqrt{y} \leq X \leq \sqrt{y})$$

になりますが、標準正規分布は原点に関して対称なので、

$$G(y) = 2P(0 \leq X \leq \sqrt{y}) = 2\int_0^{\sqrt{y}} (2\pi)^{\frac{1}{2}} e^{-\frac{x^2}{2}} dx$$

となります。ここで、$u = x^2$ となる変数変換を行うと、$dx = \frac{1}{2}u^{-\frac{1}{2}}du$ より、

$$G(y) = \int_0^y \frac{1}{\sqrt{2\pi}} u^{-\frac{1}{2}} e^{-\frac{u}{2}} du \tag{B.17}$$

となります。従って確率密度関数は B.17 を微分して、

$$g(y) = \frac{1}{\Gamma(\frac{1}{2})2^{\frac{1}{2}}} y^{-\frac{1}{2}} e^{-\frac{y}{2}} \tag{B.18}$$

となります。ここで Γ はガンマ関数で、$\Gamma(s) = \int_0^\infty x^{s-1}e^{-x}dx$、$\Gamma(\frac{1}{2}) = \frac{1}{\sqrt{\pi}}$ となることが知られています [8]。

$Y_n = X_1^2 + X_2^2 + \cdots + X_n^2$ の分布は、

$$f(x) = \frac{1}{\Gamma(\frac{n}{2})2^{\frac{n}{2}}} x^{\frac{n}{2}-1} e^{-\frac{x}{2}}, 0 < x < \infty \tag{B.19}$$

となることが知られています [8], [36]。式 B.19 に従う分布を χ^2 分布と呼び、n を自由度と呼びます。

B.3 エントロピーの期待値

互いに独立な確率変数 $\xi_1, \xi_2, \xi_3, \cdots$ を考え、各々は、p_i の確率で値 E_i $(0 \leq p_i < 1, i = 1, 2, \cdots, s,\ \sum_{i=1}^s p_i = 1)$ になるとします。n 個の塩基を考える場合、例えば $\xi_t \in \{E_1, E_2, E_3, E_4\}, 1 \leq t \leq n, s = 4, E_1 = \mathrm{a}, E_2 = \mathrm{c}, E_3 = \mathrm{g}, E_4 = \mathrm{t}$ となります。

エントロピーは次の式で定義されました (節 1.2 参照)。

$$H = H(p_1, \cdots, p_s) = -\sum_{i=1}^s p_i \log_2 p_i \tag{B.20}$$

大きさ N の標本からの B.20 の H の推定量 $\hat{H}$ は次式で与えられます。

$$\hat{H} = H(\hat{p}_1, \cdots, \hat{p}_s) = -\sum_{i=1}^s \hat{p}_i \log_2 \hat{p}_i$$

そして期待値は

$$\mathrm{E}(\hat{H}) = H - \frac{s-1}{2N} \log_2 e + O\left(\frac{1}{N^2}\right) \tag{B.21}$$

となるのです [37]。そこで順を追って式 B.21 を証明しましょう。まず H を以下のように再定義します。

$$H = H(p_1, \cdots, p_s) = -\sum_{i=1}^s p_i \log_e p_i \tag{B.22}$$

次のテイラー展開 [6]

$$f(b) = f(a) + \frac{f'(a)}{1!}(b-a) + \frac{f''(a)}{2!}(b-a)^2 + \cdots$$

$$\cdots + \frac{f^{(n-1)}(a)}{(n-1)!}(b-a)^{n-1} + \frac{f^{(n)}\{a+\theta(b-a)^n\}}{n!} \qquad \text{(B.23)}$$

$0 < \theta < 1$

と、

$$\frac{d(-p_i \log_e p_i)}{dp_i} = -1 - \log_e p_i, \quad \frac{d^k(-p_i \log_e p_i)}{dp_i^k} = (-1)^{k-1}(k-2)!p_i^{-k+1}$$

$i = 1, 2, \cdots, s \quad k \geq 2$

を使えば、以下が導かれます。

$$-\hat{p}_i \log_e \hat{p}_i = -p_i \log_e p_i - (\hat{p}_i - p_i)(1 + \log_e p_i)$$
$$-\frac{1}{2} \cdot \frac{(\hat{p}_i - p_i)^2}{p_i} + \frac{1}{6} \cdot \frac{(\hat{p}_i - p_i)^3}{p_i^2} - \frac{1}{12} \cdot \frac{(\hat{p}_i - p_i)^4}{\{p_i + \theta(\hat{p}_i - p_i)\}^3} \qquad \text{(B.24)}$$

$0 < \theta < 1$

式 B.24 の両辺を $i = 1, 2, \cdots, s$ について足すことにより、

$$-\sum_{i=1}^{s} \hat{p}_i \log_e \hat{p}_i = \hat{H} = H(\hat{p}_1, \cdots, \hat{p}_s)$$

$$= H(p_1, \cdots, p_s) - \sum_{i=1}^{s} (\hat{p}_i - p_i)(1 + \log_e p_i)$$
$$-\sum_{i=1}^{s} \frac{1}{2} \cdot \frac{(\hat{p}_i - p_i)^2}{p_i} + \sum_{i=1}^{s} \frac{1}{6} \cdot \frac{(\hat{p}_i - p_i)^3}{p_i^2} - \sum_{i=1}^{s} \frac{1}{12} \cdot \frac{(\hat{p}_i - p_i)^4}{\{p_i + \theta(\hat{p}_i - p_i)\}^3} \qquad \text{(B.25)}$$

が導かれます。また一般に以下のような関係が成立します [8]

$$\mathrm{E}(\hat{p}_i) = p_i \qquad \text{(B.26)}$$

$$\mathrm{E}((\hat{p}_i - p_i)^2) = \frac{p_i(1-p_i)}{N} \qquad \text{(B.27)}$$

式 B.26 を式 B.25 の最も右の式の第 2 項に代入すると、

$$\mathrm{E}\left(\sum_{i=1}^{s} (\hat{p}_i - p_i)(1 + \log_e p_i)\right) = \mathrm{E}\left(\sum_{i=1}^{s} (p_i - p_i)(1 + \log_e p_i)\right) = 0$$

式 B.27 を式 B.25 の最も右の式の第 3 項に代入すると、

$$\mathrm{E}\left(\sum_{i=1}^{s} \frac{1}{2} \cdot \frac{(\hat{p}_i - p_i)^2}{p_i}\right) = \frac{1}{2}\sum_{i=1}^{s} \mathrm{E}\left(\frac{(\hat{p}_i - p_i)^2}{p_i}\right) = \frac{1}{2}\sum_{i=1}^{s} \frac{p_i(1-p_i)}{Np_i} = \frac{s-1}{2N}$$

式 B.25 の最も右の式の第 4 項以降は、$O(\frac{1}{N^2})$ となるので、

$$\mathrm{E}(\hat{H}) = H - \frac{s-1}{2N} + O\left(\frac{1}{N^2}\right) \qquad \text{(B.28)}$$

となります。結局式 B.28 の両辺に $\log_2 e$ を掛けることにより $\frac{\log_2 p_i}{\log_2 e} = \log_e p_i$ を使って B.21 が導かれます。

B.4 独立性の検定

n 個の要素が種別 A では r 個のグループに分けることができ、また種別 B では c 個のグループに分けることができると仮定します。$n_{i\cdot}$ を種別 A のグループ i に属する要素の数とし、$n_{\cdot j}$ を種別 B のグループ j に属する要素の数とします。また n_{ij} を種別 A ではグループ i に属し、種別 B ではグループ j に属する要素の数とします。また、$n_{i\cdot}$、$n_{\cdot j}$、n を以下の式により定義します。

$$n_{i\cdot} = \sum_{j=1}^{c} n_{ij} \quad n_{\cdot j} = \sum_{i=1}^{r} n_{ij}$$

$$n = \sum_{i=1}^{r} n_{i\cdot} = \sum_{j=1}^{c} n_{\cdot j} = \sum_{i=1}^{r}\sum_{j=1}^{c} n_{ij} \tag{B.29}$$

同様にして、$p_{i\cdot}$ をある要素が種別 A のグループ i に属する確率、$p_{\cdot j}$ をある要素が種別 B のグループ j に属する確率とします。そして p_{ij} をある要素が種別 A ではグループ i に属し、種別 B ではグループ j に属する確率とします。

$$p_{i\cdot} = \sum_{j=1}^{c} p_{ij} \quad p_{\cdot j} = \sum_{i=1}^{r} p_{ij}$$

$$\sum_{i=1}^{r} p_{i\cdot} = \sum_{j=1}^{c} p_{\cdot j} = \sum_{i=1}^{r}\sum_{j=1}^{c} p_{ij} = 1 \tag{B.30}$$

p の尤度関数[*2)]は、

$$f(N|p) = \binom{n}{n_{11}\cdots n_{rc}} p_{11}^{n_{11}} \cdots p_{rc}^{n_{rc}} \tag{B.31}$$

ここで、帰無仮説

$$p_{ij} = p_{i\cdot}p_{\cdot j} \tag{B.32}$$

すなわち、ある要素が種別 A でグループ i に属する確率とその要素が種別 B でグループ j に属する確率は互いに独立であるかを検定する方法を考えます。

式 B.31 の最尤推定量を得るために、ラグランジュの乗数法を使います。まず式 B.31 の対数をとり、それを関数 g の中に以下のように取り入れます。

$$g(p,\lambda) = \log\binom{n}{n_{11}\cdots n_{rc}} + \sum_{i=1}^{r}\sum_{j=1}^{c} n_{ij}\log p_{ij} + \lambda(p_{11}+\cdots+p_{rc}-1) \tag{B.33}$$

但し、式 B.33 の最後の項は B.30 の制約を表します。そして式 B.33 を p_{ij} と λ で偏微分し、以下の式を解いて極値を求めます。

$$\begin{cases} \frac{\partial g}{\partial p_{ij}} = \frac{n_{ij}}{p_{ij}} - \lambda = 0 \\ \frac{\partial g}{\partial \lambda} = p_{11} + \cdots + p_{rc} - 1 = 0 \end{cases} \tag{B.34}$$

*2) 尤度関数や最尤推定量に関しては、節 C.1 参照

ここで $\frac{n_{ij}}{p_{ij}} = \lambda$ が全ての i,j に対して成立するので、$n_{ij} = \lambda p_{ij}$ の両辺を全ての i,j について足して $\lambda = n$ となり、$p_{ij} = \frac{n_{ij}}{n}$ で式 B.31 は極値となります。従って、式 B.31 の最尤推定量 $\hat{p}_{ij}$ は

$$\hat{p}_{ij} = \frac{n_{ij}}{n}$$

となります。ここで帰無仮説

$$p_{ij} = q_{ij}$$

と対立仮説

$$p_{ij} \neq q_{ij}$$

を設定すると、対数尤度比統計量 k は、

$$\begin{aligned} k &= 2\log\frac{\max_p(f(N|p))}{\max_q(f(N|q))} = 2\log\frac{\binom{n}{n_{11}\cdots n_{rc}}\left(\frac{n_{11}}{n}\right)^{n_{11}}\cdots\left(\frac{n_{rc}}{n}\right)^{n_{rc}}}{\binom{n}{n_{11}\cdots n_{rc}}q_{11}^{n_{11}}\cdots q_{rc}^{n_{rc}}} \\ &= 2\sum_{i=1}^{r}\sum_{j=1}^{c} n_{ij}\log\frac{n_{ij}}{nq_{ij}} \end{aligned} \tag{B.35}$$

と表され、k は自由度 $\dim(p)$-$\dim(q)$ の χ^2 分布に従うことが知られています[38]~[40]。$\dim(p) = (r-1)(c-1)$,$\dim(q) = 0$(q は固定) なので、k は自由度 $(r-1)(c-1)$ の chi^2 分布に従います。

$$k \sim \chi^2$$

ここで $P_{i\cdot} = \frac{n_i}{n}, P_{\cdot j} = \frac{n_j}{n}, P_{ij} = \frac{n_{ij}}{n}$ とすると、式 B.35 は以下のように書き換えられます。

$$k = 2n\sum_{i=1}^{r}\sum_{j=1}^{c} P_{ij}\log\frac{P_{ij}}{q_{ij}} \tag{B.36}$$

結局、帰無仮説を式 B.32 のようにすると、

$$k = 2n\sum_{i=1}^{r}\sum_{j=1}^{c} P_{ij}\log\frac{P_{ij}}{P_{i\cdot}P_{j\cdot}}$$

となるので、種別 A において所属するグループと、種別 B において所属するグループが独立かを調べるには、k に対して χ^2 検定を行えばいいことになります。

B.5　ラグランジュの未定乗数法

ラグランジュの未定乗数法はある制約条件のもとで、与えられた関数の極値を求める手法です。今、

$$g(x,y)=0 \tag{B.37}$$

という制約条件のもとで、$f(x,y)$ の極値およびその極値を与える点 $(x,y)=(x^0,y^0)$ を求めることを考えましょう。条件 $g(x,y)=0$ が陽関数の形で $y=y(x)$ と書くことができれば、関数 $f(x,y(x))$ が $x=x^0$ で極値を取るのでその点で、

$$\frac{df}{dx}=\frac{\partial f}{\partial x}\frac{dx}{dx}+\frac{\partial f}{\partial y}\frac{dy}{dx}=\frac{\partial f}{\partial x}+\frac{\partial f}{\partial y}\frac{dy}{dx}=0 \tag{B.38}$$

が成り立ちます[*3)]。一方条件 B.37 より、

$$dg(x,y)=\frac{\partial g}{\partial x}dx+\frac{\partial g}{\partial y}dy=0 \tag{B.39}$$

となるので、$\frac{\partial g}{\partial y}\neq 0$ のとき、式 B.39 を変形して $\frac{dy}{dx}=-\frac{\partial g}{\partial x}/\frac{\partial g}{\partial y}$ と表されます。これを式 B.38 に代入すると、$x=x^0$ で $\frac{\partial f}{\partial x}-\frac{\partial f}{\partial y}\frac{\partial g}{\partial x}/\frac{\partial g}{\partial y}=0$ が成立します。これを変形して $\frac{\partial f}{\partial x}/\frac{\partial g}{\partial x}=\frac{\partial f}{\partial y}/\frac{\partial g}{\partial y}=\lambda$ とおけば、

$$\frac{\partial f}{\partial x}-\lambda\frac{\partial g}{\partial x}=0,\quad \frac{\partial f}{\partial y}-\lambda\frac{\partial g}{\partial y}=0 \tag{B.40}$$

となります。つまり与えられた条件のもとで f が極値を持つとき、式 B.40 が成立するので、式 B.40 を満たす (x,y) は f を極値にする点の候補となります。このようにして、f の極値およびそのときの (x^0,y^0) を求める方法を、ラグランジュの未定乗数法と呼びます。より一般化すると、以下が成立します[5], [41]。

変数 $x_1,x_2,\cdots,x_n$ は次の $m(<n)$ 個の等式で制約されているものとします。

$$g_i(x_1,x_2,\cdots,x_n)=0\quad (i=1,2,\cdots,m) \tag{B.41}$$

このとき、関数 $f(x_1,x_2,\cdots,x_n)$ が点 $(x_1^0,x_2^0,\cdots,x_n^0)$ で最大（または最小）となるための必要条件は、以下のようになります。

$$\frac{\partial f}{\partial x_1}+\sum_{i=1}^{m}\lambda_i\frac{\partial g_i}{\partial x_1}=0,\frac{\partial f}{\partial x_2}+\sum_{i=1}^{m}\lambda_i\frac{\partial g_i}{\partial x_2}=0,\cdots,\frac{\partial f}{\partial x_n}+\sum_{i=1}^{m}\lambda_i\frac{\partial g_i}{\partial x_n}=0 \tag{B.42}$$

B.6　対応分析と相関係数の最大化

今、$f_{..}$ 個の要素が種別 A では m 個のグループのうちの必ず 1 つに分類され、また同時に種別 B では n 個のグループのうちの必ず 1 つに分類されるとします。そして、種別 A ではグループ $i(1\leq i\leq m)$ に属し、種別 B ではグルー

*3)　これを f の全微分と呼ぶ[6]。

プ $j(1 \leq j \leq n)$ に属する要素が f_{ij} 個あると仮定します。ここで、節 6.5 と同様、行列 F、S、C を定義します。すなわち、

$$F = (f_{ij}), \quad S = (s_{ij}), \quad C = (c_{ij})$$

$$s_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ f_{i\cdot} & \text{if } i = j \end{cases}, \quad c_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ f_{\cdot j} & \text{if } i = j \end{cases}$$

$$f_{i\cdot} = \sum_{l=1}^{n} f_{il}, \quad f_{\cdot j} = \sum_{k=1}^{m} f_{kj}, \quad f_{\cdot\cdot} = \sum_{k=1}^{m}\sum_{l=1}^{n} f_{kl}$$

とします。

種別 A におけるそれぞれのグループ i にはそれぞれ x_i を、種別 B における、それぞれのグループ j にはそれぞれ y_j という値を対応させることを考えましょう。ここで x_i, y_j の平均を 0、すなわち

$$\sum_{k=1}^{m} x_k f_{k\cdot} = 0 \quad \sum_{l=1}^{n} y_l f_{\cdot l} = 0 \tag{B.43}$$

とします。また x と y の分散を 1、すなわち、

$$\frac{x^t S x}{f_{\cdot\cdot}} = 1 \quad \frac{y^t C y}{f_{\cdot\cdot}} = 1 \tag{B.44}$$

とします。式 B.43、B.44 の制約の中で x と y の相関係数 r を最大にする x、y はどのようになるか、以下求めてゆきましょう [20]。

まず相関係数 r は、

$$r = \frac{x^t F y}{\sqrt{x^t S x y^t C y}} = \frac{x^t F y}{f_{\cdot\cdot}}$$

となりますが、分母は定数となるため、$x^t F y$ を最大にすればいいことになります。するとラグランジュの未乗数法により r を最大にするには、

$$f(x, y) = x^t F y - \frac{\lambda}{2}(x^t S x - f_{\cdot\cdot}) - \frac{\mu}{2}(y^t C y - f_{\cdot\cdot}) \tag{B.45}$$

を最大にすればいいことになります。式 B.45 を x、y で偏微分して、

$$\frac{\partial f}{\partial x} = F y - \lambda S x = 0 \tag{B.46}$$

$$\frac{\partial f}{\partial y} = F^t x - \mu C y = 0 \tag{B.47}$$

を得ます。式 B.46 に左側から x^t を掛けると、

$$x^t F y - \lambda x^t S x = x^t F y - f_{\cdot\cdot}\lambda = 0 \tag{B.48}$$

となります。同様に式 B.47 の左側から y^t を掛けて、

$$y^t F^t x - \mu y^t C y = y^t F^t - f_{\cdot\cdot}\mu = x^t F y - f_{\cdot\cdot}\mu = 0 \tag{B.49}$$

を得ます。すると式B.48とB.49より$\lambda = \mu$となり、かつ式B.48より$\lambda = \frac{x^t F y}{f_{..}}$となる。よって、$\lambda$が$r$の最大値となります。

式B.47のμをλに置き換え、移項することにより、$\lambda y = C^{-1}F^t x$となります。式B.46にλを掛けた後にこれを代入して、

$$FC^{-1}F^t x - \lambda^2 S x = 0 \tag{B.50}$$

となります。さらに式B.50に左側から$S^{-\frac{1}{2}}$を掛け、$E = S^{-\frac{1}{2}}S^{\frac{1}{2}}$を使うことにより、

$$S^{-\frac{1}{2}}FC^{-1}F^t S^{-\frac{1}{2}}(S^{\frac{1}{2}}x) - \lambda^2(S^{\frac{1}{2}}x) = 0$$

すなわち、

$$(S^{-\frac{1}{2}}FC^{-1}F^t S^{-\frac{1}{2}} - \lambda^2 E)S^{\frac{1}{2}}x = 0 \tag{B.51}$$

を満たすようなxがrを極大にすることが分かりました。yは$y = \frac{1}{\lambda}C^{-1}F^t x$より求めることができます。

式6.8を式6.14を用いて変形すると、

$$\frac{1}{\sqrt{f_{..}}}(S^{-\frac{1}{2}}FC^{-1}F^t S^{-\frac{1}{2}} - d_{ii}^2 E)S^{\frac{1}{2}}x_i = 0 \tag{B.52}$$

となります。但しEはn次元の単位行列です。式B.52の両辺に$\sqrt{f_{..}}$を掛ければ式B.51とB.52は同じ形の式になります。従って式B.52を満たすxおよびd_{ii}(式B.51ではλ)が式B.51も満たすことが分かります。結局対応分析は、種別Aの各グループと種別Bの各グループにA、B間の相関が最大になるように数値を与える手法であることが分かります。

なお、$\lambda = 1, x = 1, y = 1$が式B.46、B.47の解となっており、対応分析をするとこれが次元の1つ(最初の次元)として出てくることが分かります。

付録 C
EM アルゴリズム

C.1 EM アルゴリズムの概要

EM アルゴリズムは最尤推定量を数値的に求めるアルゴリズムです。今、パラメータ θ によってきまる確率関数を $P(x|\theta)$ とします。観測値 x が与えられたとき、母数 θ をどのように推定すればいいでしょうか？

最尤推定法では観測値 x を定数として、$P(x|\theta)$ が最大になるような θ を真の θ の推定量とします。この推定量すなわち、

$$\theta_{\max} = \mathrm{argmax}_\theta P(x|\theta) = \mathrm{argmax}_\theta \log P(x|\theta) \tag{C.1}$$

を最尤推定量といいます。最尤推定量は一致性や漸近正規性など統計解析をする上で有用な性質を持っています [8], [42]。全ての確率変数が観測可能なケースでは多くの場合、最尤推定量は解析的に求めることが可能です。しかし、確率変数の一部が観測不能な場合、最尤推定量を解析的に求めるのは多くの場合、困難です。そこで EM アルゴリズムではこれを数値的に求めます。

なお θ を関数への入力と考え、$P(x|\theta)$ を θ が真の θ の推定量としての尤もらしさを表す関数と考えるとき、$P(x|\theta)$ を尤度関数、その値自体を尤度といいます。

観測可能な確率変数を x、観測不能な確率変数を y とすると、

$$\mathrm{argmax}_\theta \log P(x|\theta) = \mathrm{argmax}_\theta \left(\log \sum_y P(x, y|\theta) \right) \tag{C.2}$$

となります。また $P(x, y|\theta) = P(y|x, \theta)P(x|\theta)$ を移項して両辺の対数をとると、

$$\log P(x|\theta) = \log P(x, y|\theta) - \log P(y|x, \theta) \tag{C.3}$$

となります。θ のとりあえずの推定量を θ^t として、式 C.3 の両辺に $P(y|x, \theta^t)$ を掛けて、y について和をとると、

$$左辺 = \sum_y P(y|x,\theta^t)\log P(x|\theta) = \log P(x|\theta)$$

$$右辺 = \sum_y P(y|x,\theta^t)\log P(x,y|\theta) - \sum_y P(y|x,\theta^t)\log P(y|x,\theta)$$

従って、

$$\log P(x|\theta) = \sum_y P(y|x,\theta^t)\log P(x,y|\theta) - \sum_y P(y|x,\theta^t)\log P(y|x,\theta) \quad \text{(C.4)}$$

となります。ここで、

$$Q(\theta|\theta^t) = \sum_y P(y|x,\theta^t)\log P(x,y|\theta) \quad \text{(C.5)}$$

とおいて、式 C.4 の両辺から $\log P(x|\theta^t)$ を引くと、

$$\log P(x|\theta) - \log P(x|\theta^t) = Q(\theta|\theta^t) - Q(\theta^t|\theta^t) + \sum_y P(y|x,\theta^t)\log\frac{P(y|x,\theta^t)}{P(y|x,\theta)} \quad \text{(C.6)}$$

となります。式 C.6 の右辺の $\sum_y P(y|x,\theta^t)\log\frac{P(y|x,\theta^t)}{P(y|x,\theta)}$ は増加情報量であり、$\sum_y P(y|x,\theta^t)\log\frac{P(y|x,\theta^t)}{P(y|x,\theta)} \geq 0$ となるので (節 1.3 参照)、

$$\log P(x|\theta) - \log P(x|\theta^t) \geq Q(\theta|\theta^t) - Q(\theta^t|\theta^t) \quad \text{(C.7)}$$

が成立します。ここで式 C.7 の θ に θ^{t+1} を代入すると、$Q(\theta^{t+1}|\theta^t) \geq Q(\theta^t|\theta^t)$ が成立すれば、

$$\log P(x|\theta^{t+1}) - \log P(x|\theta^t) \geq Q(\theta^{t+1}|\theta^t) - Q(\theta^t|\theta^t) \geq 0 \quad \text{(C.8)}$$

となります。$\theta^{t+1} = \mathrm{argmax}_\theta Q(\theta|\theta^t)$ とおけば、$Q(\theta^{t+1}|\theta^t) \geq Q(\theta^t|\theta^t)$ が成立するので、θ^t を θ^{t+1} と置き換えることによって $\log P(x|\theta^t)$ が $Q(\theta^{t+1}|\theta^t) - Q(\theta^t|\theta^t)$ 以上上昇することになります。

また、$\theta^{t+2} = \mathrm{argmax}_\theta Q(\theta|\theta^{t+1})$ とおけば、$Q(\theta^{t+2}|\theta^{t+1}) \geq Q(\theta^{t+1}|\theta^{t+1})$ となり、$\log P(x|\theta^{t+2}) - \log P(x|\theta^{t+1}) \geq Q(\theta^{t+2}|\theta^{t+1}) - Q(\theta^{t+1}|\theta^{t+1}) \geq 0$ が成立します。よって、θ^{t+1} を θ^{t+2} と置き換えることによって $\log P(x|\theta^{t+1})$ が $Q(\theta^{t+2}|\theta^{t+1}) - Q(\theta^{t+1}|\theta^{t+1})$ 以上上昇することになります。

これを繰り返して、θ^t から θ^{t+1}、θ^{t+1} から θ^{t+2}、$\cdots$ を求めていくと、$P(x|\theta^{t+})$ はどんどん上昇していきます。このようにして数値的に $P(x|\theta^{t+})$ を増していくのが EM アルゴリズムであり、以下のようにまとめることができます。

E-Step $Q(\theta|\theta^t) = \sum_y P(y|x,\theta^t)\log P(x,y|\theta) = E_y[\log P(x,y|\theta)|x,\theta^t]$ の計算。観測値 x と現在の推定量 θ^t を用いて、観測不能データ y に関する $\log P(x,y|\theta)$ の平均値を求める

M-Step $Q(\theta|\theta^t)$ の θ に関する最大化。$\theta^{t+1} = \mathrm{argmax}_\theta Q(\theta|\theta^t)$ を求め、これを新たな推定量とする。

C.2 K 平均アルゴリズムの導出

では、EM アルゴリズムを用いて K 平均アルゴリズムを導出してみましょう。確率変数 $x_i(1 \leq x \leq m)$ が平均 $\mu_1, \mu_2, \cdots$ または μ_k のいずれかの正規分布に従うとします。但し分散はいずれも σ^2 とします。

また変数 y_{ij} を

$$y_{ij} = \begin{cases} 1 & \text{if } x_i \sim N(\mu_j, \sigma^2) \\ 0 & \text{otherwise} \end{cases}$$

と定義します。クラスタリング対象となる要素 i の確率変数は $(x_i, y_{i1}, y_{i2}, \cdots, y_{ik})$ で表されることになりますが、x_i が観測可能な確率変数、$y_{i1}, \cdots, y_{ik}$ が観測不能な確率変数となります。i が k 個の正規分布のうち、どこに従う確率も $\frac{1}{k}$ とすると、$(x_i, y_{i1}, y_{i2}, \cdots, y_{ik})$ の分布を表す確率関数は式 B.15 をもとにして、

$$P(x_i, y_{i1}, y_{i2}, \cdots, y_{ik}|\mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}\sum_{j=1}^{k} y_{ij}(x_i-\mu_i)^2} \tag{C.9}$$

となります。但し、μ は $\mu_1, \cdots, \mu_k$ を表します。式 C.9 を m 個の要素についての積に書き直し、さらに両辺について自然対数をとると、

$$\log_e P(x, y|\mu) = \log_e \prod_{i=1}^{m} P(x_i, y_{i1}, y_{i2}, \cdots, y_{ik}|\mu)$$

$$= \sum_{i=1}^{m} \log_e P(x_i, y_{i1}, y_{i2}, \cdots, y_{ik}|\mu)$$

$$= \sum_{i=1}^{m} \left(\log_e \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} y_{ij}(x_i - \mu_j)^2 \right) \tag{C.10}$$

と表すことができます。

従って E-Step では、

$$Q(\mu|\mu^t) = \sum_{y} P(y|x, \mu^t) \log_e P(x, y|\mu)$$

$$= E_y[\log_e P(x, y|\mu)|x, \mu^t] = E_y \left[\sum_{i=1}^{m} \left(\log_e \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} y_{ij}(x_i - \mu_j)^2 \right) |x, \mu^t \right]$$

$$= \sum_{i=1}^{m} \left(\log_e \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[y_{ij}|x, \mu^t](x_i - \mu_j)^2 \right)$$

を計算します。さらに E の部分は、

$$E[y_{ij}|x, \mu^t] = \sum_{y_{ij}=0,1} P(y_{ij}|x_i, \mu^t) y_{ij} = P(y_{ij} = 1|x_i, \mu^t)$$

$$= \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x_i-\mu_j^t)^2}}{\sum_{v=1}^{k} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x_i-\mu_v^t)^2}} = \frac{e^{-\frac{1}{2\sigma^2}(x_i-\mu_j^t)^2}}{\sum_{v=1}^{k} e^{-\frac{1}{2\sigma^2}(x_i-\mu_v^t)^2}}$$

となります。

M-Step では、

$$\mu^{t+1} = \mathrm{argmax}_\mu Q(\mu|\mu^t)$$

$$= \mathrm{argmax}_\mu \sum_{i=1}^{m} \left(\log_e \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{j=1}^{k} E[y_{ij}|x,\mu^t](x_i-\mu_j)^2 \right)$$

$$= \mathrm{argmin}_\mu \sum_{i=1}^{m} \sum_{j=1}^{k} E[y_{ij}|x,\mu^t](x_i-\mu_j)^2 \qquad \text{(C.11)}$$

を求めます。そのためには式 C.11 を各クラスター v の平均 μ_v で偏微分して、

$$\frac{\partial}{\partial \mu_v} \sum_{i=1}^{m} \sum_{j=1}^{k} E[y_{ij}|x,\mu^t](x_i-\mu_j)^2 = \frac{\partial}{\partial \mu_v} \sum_{i=1}^{m} E[y_{iv}|x,\mu^t](x_i-\mu_v)^2$$

$$= -2 \sum_{i=1}^{m} E[y_{iv}|x,\mu^t](x_i-\mu_v) = 0$$

とおけば、$\mu_v = \frac{\sum_{i=1}^{m} E[y_{iv}|x,\mu^t] x_i}{\sum_{i=1}^{m} E[y_{iv}|x,\mu^t]}$ となるので、

$$\mu_v^{t+1} = \frac{\sum_{i=1}^{m} E[y_{iv}|x,\mu^t] x_i}{\sum_{i=1}^{m} E[y_{iv}|x,\mu^t]}$$

が M-Step となります。

節 5.4 では $E[y_{iv}|x,\mu^t]$ の代わりに、$N_b[y_{iv}|x,\mu^t]$ を使っており、その定義は以下の通りです。

$$N_b[y_{ij}|x,\mu^t] = \begin{cases} 1 & \text{if } j = \mathrm{argmin}_{j'}(x_i-\mu_{j'}^t)^2 \\ 0 & \text{otherwise} \end{cases}$$

C.3 隠れマルコフモデルのパラメータ推定

では次に隠れマルコフモデルにおいて、配列群 x が観測されたときの、$P(x|\theta)$ における θ の最尤推定量を求めることを考えましょう。まず式 C.5 で y を観測不能な変数である π に置き換えて、

$$Q(\theta|\theta^t) = \sum_{\pi} P(\pi|x,\theta^t) \log P(x,\pi|\theta) \qquad \text{(C.12)}$$

とします。パラメータ θ を持つ図 C.1 のような隠れマルコフモデルにおいて、経路 $\pi = (\mathrm{start}, \pi_1, \cdots, \pi_L, \mathrm{end})$ を通り、配列 x が出力される確率は、

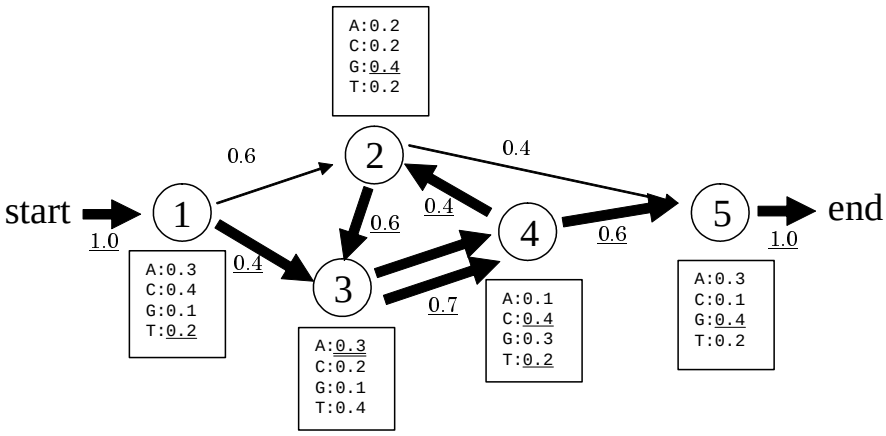


図 C.1　隠れマルコフモデル上の通過経路および出力記号の例

”TACGATG”が出力されたときに通った経路の例。通過した経路の矢印を太く描いてある。また 2 度通った経路には矢印を 2 本描き、同じ状態より 2 度出力された記号には 2 本の下線を引いた。

$$P(x,\pi|\theta) = t(\text{start} \to \pi_1)\prod_{i=1}^{L} e_{\pi_i}(x_i)t(\pi_i \to \pi_{i+1}) \tag{C.13}$$

となります。但し L は配列の長さ＝通過した記号を出力する状態の数で、$\pi_{L+1} = \text{end}$ とします。

これは通った状態の順序に着目した定式化の方法ですが、これを順序には特にこだわらず、各状態および状態遷移を使用した回数に着目した定式化を行ってみましょう。図 C.1 には、塩基配列"TACGATG"が出力されたときに通った経路を示してあります。式 C.13 を用いて、図 C.1 のような経路を通ってこのような塩基配列が出力される確率を表すと、

$$e_1(\mathrm{T})t(1\to 3)e_3(\mathrm{A})t(3\to 4)e_4(\mathrm{C})t(4\to 2)e_2(\mathrm{G})t(2\to 3)e_3(\mathrm{A})t(3\to 4)$$
$$\times e_4(\mathrm{T})t(4\to 5)e_5(\mathrm{G})$$

ですが、これを状態やその遷移ごとにまとめると、

$$e_1(\mathrm{T})^1 e_2(\mathrm{G})^1 e_3(\mathrm{A})^2 e_4(\mathrm{C})^1 t(1\to 3)^1 t(2\to 3)^1 t(3\to 4)^2 t(4\to 2)^1$$

となります。但し e のべき乗項はその状態においてその塩基が出力された回数、t のべき乗項はその状態遷移が使われた回数です。これを一般化して、今経路 π 上において、状態 k から状態 l への遷移が使われる回数を $N_{t(k\to l)}(\pi)$、塩基 b が状態 k において観測される回数を $N_{e_k(b,\pi)}$ とします。すると、式 C.13 は以下のように書き直せます。

$$P(x,\pi|\theta) = \prod_{k=1}^{K}\prod_{b}[e_k(b)_{|\theta}]^{N_{e_k}(b,\pi)} \prod_{k=0}^{K}\prod_{l=1}^{K+1}[t(k\to l)_{|\theta}]^{N_{t(k\to l)}(\pi)} \tag{C.14}$$

但し、K は start、end を除いた状態の数です。また塩基を出力する各状態には状態番号 1〜K が付けられ、start は状態番号 0 とします。式 C.14 を式 C.12 に代入すれば、

$$Q(\theta|\theta^t) = \sum_{\pi} P(\pi|x,\theta^t)\times$$
$$\left[\sum_{k=1}^{K}\sum_{b} N_{e_k(b,\pi)}\log e_k(b)_{|\theta} + \sum_{k=0}^{K}\sum_{l=1}^{K+1} N_{t(k\to l)}(\pi)\log t(k\to l)_{|\theta}\right] \tag{C.15}$$

となります。

ここで期待値の定義を思い出して下さい。確率変数 X が $p_1, p_2, \cdots, p_n$(但し $\sum_{i=1}^{n} p_i = 1$) の確率で、値 $x_1, x_2, \cdots x_n$ を取るとき ($P(X = x_i) = p_i$)、$E(X) = \sum_{i=1}^{n} p_i x_i$ を確率変数 X の期待値と呼んでいました。$N_{t(k\to l)}(\pi)$ を確率変数、$P(\pi|x,\theta^t)$ をその確率とすれば期待値を表す式 3.14 は、

$$N_{t(k\to l)|\theta^t} = \sum_{\pi} P(\pi|x,\theta^t)N_{t(k\to l)}(\pi) \tag{C.16}$$

と書き直すことができます。同様に、$N_{e_k}(b,\pi)$ を確率変数、$P(\pi|x,\theta^t)$ をその確率とすれば、式 3.17 は

$$N_{e_k(b)|\theta^t} = \sum_{\pi} P(\pi|x,\theta^t) N_{e_k}(b,\pi) \tag{C.17}$$

と書き直すことができます。式 C.16,C.17 を式 C.15 に代入すると、

$$Q(\theta|\theta^t) = \sum_{k=1}^{K}\sum_{b} N_{e_k(b)|\theta^t} \log e_k(b)_{|\theta} + \sum_{k=0}^{K}\sum_{l=1}^{K+1} N_{t(k\to l)|\theta^t} \log t(k\to l)_{|\theta} \tag{C.18}$$

となります。さて、式 C.18 を最大化するような $e_k(b)_{|\theta}$ および $t(k\to l)_{|\theta}$ を求めましょう。まず、右辺第 1 項の最大化から考えると、$e_k^0(b) = \frac{N_{e_k(b)|\theta^t}}{\sum_{b'} N_{e_k(b')|\theta^t}}$ とおいて、式 C.18 の第 1 項の $e_k(b)$ が $e_k^0(b)$ の場合からそれ以外の場合 ($e_k(b) = e_k^1(b)$ とおく) を引くと、

$$\begin{aligned}
&\sum_{k=1}^{K}\sum_{b} N_{e_k(b)|\theta^t} \log e_k^0(b) - \sum_{k=1}^{K}\sum_{b} N_{e_k(b)|\theta^t} \log e_k^1(b) \\
&= \sum_{k=1}^{K}\sum_{b} N_{e_k(b)|\theta^t} \log \frac{e_k^0(b)}{e_k^1(b)} = \sum_{k=1}^{K}\left(\sum_{b'} N_{e_k(b')|\theta^t}\right)\sum_{b} e_k^0(b) \log \frac{e_k^0(b)}{e_k^1(b)}
\end{aligned} \tag{C.19}$$

となります。$\sum_b e_k^0(b) \log \frac{e_k^0(b)}{e_k^1(b)}$ は増加情報量であり 0 以上なので (節 1.3 参照)、式 C.19 は $e_k(b) = e_k^0(b)$ のときに最大となります。

次に式 C.18 の第 2 項の最大化を考えましょう。まず、$t^0(k\to l) = \frac{N_{t(k\to l)|\theta^t}}{\sum_{l'} N_{t(k\to l')|\theta^t}}$ とおき、式 C.18 の第 2 項の $t(k\to l)$ が $t^0(k\to l)$ の場合からそれ以外の場合 ($t(k\to l) = t^1(k\to l)$ とおく) を引くと、

$$\begin{aligned}
&\sum_{k=0}^{K}\sum_{l=1}^{K+1} N_{t(k\to l)|\theta^t} \log t^0(k\to l) - \sum_{k=0}^{K}\sum_{l=1}^{K+1} N_{t(k\to l)|\theta^t} \log t^1(k\to l) \\
&= \sum_{k=0}^{K}\sum_{l=1}^{K+1} N_{t(k\to l)|\theta^t} \log \frac{t^0(k\to l)}{t^1(k\to l)} \\
&= \sum_{k=0}^{K}\sum_{l=1}^{K+1}\left(\sum_{l'} N_{t(k\to l')|\theta}\right) t^0(k\to l) \log \frac{t^0(k\to l)}{t^1(k\to l)}
\end{aligned} \tag{C.20}$$

となります。$\sum_l t^0(k\to l) \log \frac{t^0(k\to l)}{t^1(k\to l)}$ は増加情報量であり 0 以上となるので、式 C.20 は $t^0(k\to l) = t^1(k\to l)$ のときに最大値となります。以上の結果より式 C.18 は $e_k(b) = e_k^0(b)$、$t(k\to l) = t^0(k\to l)$ のときに最大となることが導かれました。

従って、隠れマルコフモデルのパラメータ θ を推定するための EM アルゴリズムは以下のようになります。

E-Step 式 C.18 の計算。現在の推定量 θ^t を用い、$t(k\to l)_{|\theta^t}$(式 3.14 を使う) および $N_{e_k(b)_|\theta^t}$(式 3.17 を使う) を計算する。

M-Stem 式 C.18 の最大化。

$$t(k \to l)_{|\theta^{t+1}} = \frac{N_{t(k\to l)|\theta^t}}{\sum_{l'} N_{t(k\to l')|\theta^t}}, \quad e_k(b)_{|\theta^{t+1}} = \frac{N_{e_k|\theta^t}}{\sum_{b'} N_{e_k(b')|\theta^t}}$$

で次の推定量 θ^{t+1} を求める。

参考文献

[1] 田村隆明, 新転写制御のメカニズム, 羊土社, 2000

[2] Shine J, Dalgarno L, The 3'-terminal sequence of Escherichia coli 16S ribosomal RNA: complementarity to nonsense triplets and ribosome binding sites, Proc Natl Acad Sci U S A. 71(4):1342-6, 1974

[3] Osada Y, Saito R, Tomita M, Analysis of base-pairing potentials between 16S rRNA and 5' UTR for translation initiation in various prokaryotes, Bioinformatics, 15(7-8):578-81, 1999

[4] Sazuka T, Ohara O., Sequence features surrounding the translation initiation sites assigned on the genome sequence of Synechocystis sp. strain PCC6803 by amino-terminal protein sequencing, DNA Res. 3(4):225-32, 1996

[5] 島田良作, わかる情報理論, 日新出版, 1982

[6] 小田原宏行, 微分積分学入門, 北樹出版, 1986

[7] Schneider TD, Stormo GD, Gold L, Ehrenfeucht A, Information content of binding sites on nucleotide sequences, J Mol Biol. 188(3):415-31, 1986

[8] 稲垣宣生, 数理統計学, 裳華房, 1990

[9] Mount DW 著、岡崎康司、坊農秀雅監訳，バイオインフォマティクス，メディカルサイエンスインターナショナル，2002.

[10] Smith TF, Waterman MS, Identification of common molecular subsequences, J Mol Biol. 147(1):195-7,1981

[11] Durbin, R, Eddy, SR, Krogh, A, Mitchison, G 著, 阿久津達也, 浅井潔, 矢田哲士訳, バイオインフォマティクス-確率モデルによる遺伝子配列解析-, 医学出版, 2001

[12] Numata K, Kanai A, Saito R, Kondo S, Adachi J, Wilming LG, Hume DA, Hayashizaki Y, Tomita M; RIKEN GER Group; GSL Members, Identification of putative noncoding RNAs among the RIKEN mouse full-length cDNA collection, Genome Res. 13(6B):1301-6, 2003

[13] Setubal, JC, Meidanis, J 著, 五條堀孝監訳, 遠藤俊徳訳, 分子生物学のためのバイオインフォマティクス入門, 共立出版, 2001

[14] 白井道雄, 入門物理化学, 実教出版, 1978

[15] Zuker M, Stiegler P, Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information, Nucleic Acids Res. 9(1):133-48, 1981

[16] 中村春木, 中井謙太, バイオテクノロジーのためのコンピュータ入門, コロナ社, 1995

[17] DeRisi JL, Iyer VR, Brown PO, Exploring the metabolic and genetic control of gene expression on a genomic scale, Science 278(5338):680-686, 1997

[18] Hughes TR, Marton MJ, Jones AR, Roberts CJ, Stoughton R, Armour CD, Bennett HA, Coffey E, Dai H, He YD, Kidd MJ, King AM, Meyer MR, Slade D, Lum PY, Stepaniants SB, Shoemaker DD, Gachotte D, Chakraburtty K, Simon J, Bard M, Friend SH, Functional discovery via a compendium of expression profiles, Cell, 102(1):109-26,2000

[19] Eisen MB, Spellman PT, Brown PO, Botstein D, Cluster analysis and display of genome-wide expression patterns, Proc Natl Acad Sci U S A. 95(25):14863-14868, 1998

[20] 藤沢偉作, 楽しく学べる多変量解析, 現代数学社, 1985

[21] 田中豊, 脇本和昌著, 多変量統計解析法, 現代数学社, 1983

[22] Mitchell, TM, Machine Learning, WBC/McGraw-Hill, 1997

[23] Sharp PM, Li WH, An evolutionary perspective on synonymous codon usage in unicellular organisms, J Mol Evol. 24(1-2):28-38,1986

[24] Shields DC, Sharp PM, Synonymous codon usage in *Bacillus subtilis* reflects both translational selection and mutational biases, Nucleic Acids Res. 15(19):8023-40,1987

[25] Sharp PM, Li WH, The codon Adaptation Index–a measure of directional synonymous codon usage bias, and its potential applications, Nucleic Acids Res. 15(3):1281-95,1987

[26] 吉原健一, 寺田敏司, 現代線形代数学入門, 学術図書出版社, 1991

[27] 舟尾暢男, The R Tips データ解析環境 R の基本技・グラフィックス活用術, 九天社, 2005

[28] Duret L, Evolution of synonymous codon usage in metazoans, Curr Opin Genet Dev. 12(6):640-649, 2002

[29] Muto A, Osawa S, The guanine and cytosine content of genomic DNA and bacterial evolution, Proc Natl Acad Sci U S A. 84(1):166-169,1987

[30] Perriere, G., Thioulouse, J. Use and misuse of correspondence analysis in codon usage studies. Nucleic Acids Res. 30(20):4548-55,2002

[31] Weller SC, Romney AK, Metric Scaling Correspondence Analysis, SAGE Publications Inc., 1990

[32] デイホフ J 著, 桂井浩訳, ニューラルネットワークアーキテクチャー入門, 森北出版, 1992

[33] ビール R, ジャクソン T 著, 八名和夫監訳, ニューラルコンピューティング入門, 海文堂出版, 1993

[34] 柴田文明, 理工系の基礎数学 7 確率・統計, 岩波書店, 1996

[35] 小針あき宏, 確率・統計入門 岩波書店, 1973

[36] 岩田暁一, 経営分析のための統計的方法, 東洋経済新報社, 1983

[37] Basharin, G.P., On a statistical estimate for the entropy of a sequence of independent random variables. Theory Probability Appl. 4: 333-336, 1959

[38] Wilks, SS, Mathematical Statistics, John Wiley, & Sons, 1962

[39] ウィルクス SS 著, 田中英之, 岩本誠一訳, 数理統計学 1 , 東京図書出版, 1972

[40] ウィルクス SS 著, 田中英之, 岩本誠一訳, 数理統計学 2 , 東京図書出版, 1972

[41] 笠原皓司, 微分積分学, サイエンス社, 1985

[42] 東京大学教養学部統計学教室編, 自然科学の統計学, 東京大学出版会, 1992