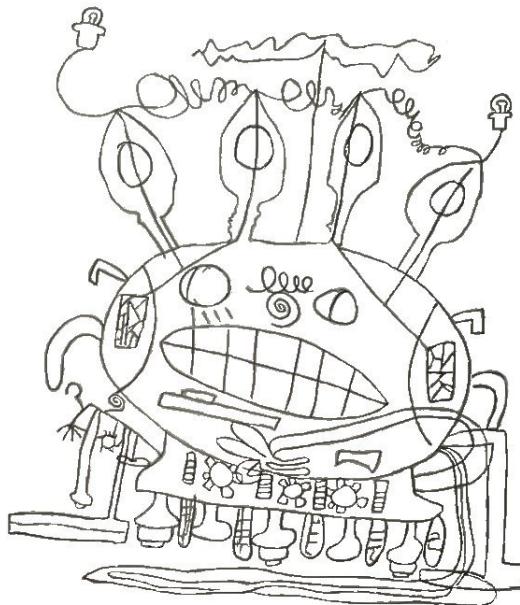


# ロボット知能 研究ノート



表紙絵：A. Honda

# 目次



# 図 目 次



# 第I部

## 理論的な側面



ここには、ロボット知能を考える上で基本的に必要になる事柄をまとめた。

反応行動のための単純な感覚運動写像を中心に、知能的な行動の創発という考え方を紹介する。

また、反応行動の基本として系が振動する場合について、基礎数学を用いた解析的な議論と数値解析的な方法も紹介する。時間遅れが存在しない場合、系が不安定になることは理論的にはありえず、せいぜい振動するだけであることを示す。

いっぽう、飛行ロボットの様な行動の特徴的な時間長さがミリ秒単位となるような、系の安定性を考える場合には、身体性がもつ時間遅れがもたらす影響が無視できない。むしろその行動を大きく左右する例を紹介する。反応の時間遅れは、系を大きく不安定にすることを示す。



# 第1章 構成論的科学としてのロボット知能

ロボット知能の研究とは、簡単に言えば人間の持っているような知能を創って、ロボットにそれをもたせようという試みと言えるであろう。似たような言葉として、人工知能というものがある。たとえばよく知られている例をあげると、コンピューターが将棋のプロ棋士に勝ったという話が話題になることがあることはご存知だろう。ここでいうコンピューターというのは、高速な演算処理をすることが可能な計算機ハードウェアと、そこにインストールされた、実際に将棋をさすプログラム（ソフトウェア）双方を一体としてコンピューターと呼んでいる。将棋プログラムは人工知能の典型的な例である。

指し手の数は膨大であるが、有限であることには違いない。巧妙に最善手を探し出すアルゴリズムを考えだされれば、記憶力も計算力も人間より桁外れの能力をもつコンピューターは人間の棋士においつき、いずれ追い越してしまうであろう。将棋のような枠組みがはっきりと定義されている問題については、人工知能のエキスパートシステムはかなりの性能を発揮することが出来るのである。このような能力も、もちろん知能であることには違いない。

われわれ人間が遭遇する状況は、このように枠組みがはっきりしている場合ばかりとは限らない。むしろ、そうではない場合がほとんどであろう。判断の条件や状況は刻一刻と変化しているし、どこまで判断の条件に含めて考えればよいのかも曖昧である。そのような状況でも、われわれは何がしかの判断を下し、適切と思われる行動を起こしている。

知能がどのように構成されていて、どのようなメカニズムの下ではた

らいているのか、はっきりわかっていれば、知能を創ることが可能かもしれない。しかし、それらがはっきり解明されていないばかりか、知能とは何か、どういうことなのかさえはっきりと定義できないのが現状である。

## 構成論的科学

そこで、最も知能的な存在であると考えられる人間を、今までの科学が用いてきた分析的な手法で研究しようとするのは、自然な考え方かもしれない。分析的な手法とは、いわば対象を徹底的に細かく細分化していき、それらの細分化された構成要素の働きの総体として、元の対象を理解しようとする方法である。例えば、物質の性質を、原子や分子が多数集まった総体として理解したり、半導体のように活用したりしようとする。このような物質科学とよばれる分野がもっともその典型的な例であろう。

ところが、知能の研究において分析的手法を用いるということは、おのずと脳を細分化して、その生物学的な性質や、電気的な機能などを調べることになる。このような分析的な努力は多くの研究者によって進められている。しかし、物質科学と同様な分析的な手法を用いることが困難であるということは、容易に想像できるであろう。たとえば、物質科学の場合のようにX線を照射して構造を調べたり電子顕微鏡でのぞいたりすることが、生きた脳に対して行えないということである。もし、行つたとしても、それらの外部刺激の影響が強すぎて、知能の働いている現場をとらえることができないということも考えられる。

そこで、構成論的科学として研究を進める方法を考える。まず「知能」を構成して、つまり、創って、それを実環境の中で行動させる。ここで、われわれ人間などが持っているような生物の知能と区別するために、構成論的に創ろうとする知能の方を「知能」と表した。「知能」はコンピューター内のいわばソフトウェアであるから、かたちとしては、その中にプログラムとして作り込むだけでも十分と思われるかもしれない。しかし、

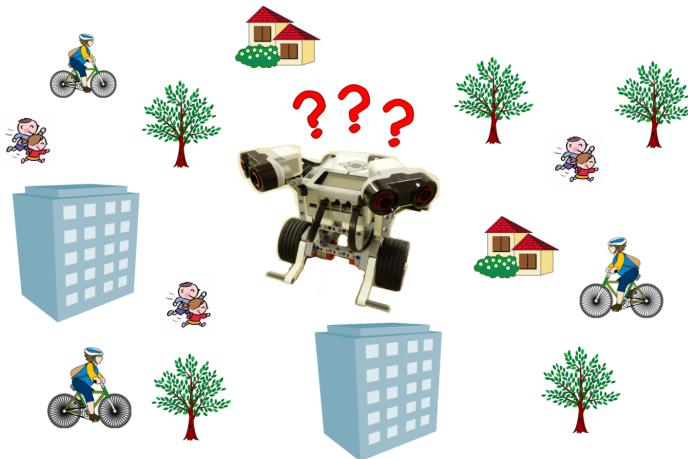


図 1.1: 環境の中で行動するロボット

構成論的科学としてのロボット知能の研究においては、「知能」と「身体」をもったロボットが変化に富んだ実環境の中で行動することが必要であると考える（図 1.1 参照）。もちろん、その「知能」で多様な実環境の中を行動するのに十分であるとは考えられない。生物であれば、そのような知能をもった生物は十分環境に適応していないということで、淘汰されるであろう。すなわち実環境に適した知能（行動原理）をもった生物だけが生き残る。

いっぽう、ロボットの場合にはその「知能」を再構築することによって同じ身体をもつロボットを再利用することができる。構成論的科学とは、いばば「知能」と「身体」を最構成しながら、生物がたどった進化の過程を再現することを通じて、知能を理解しあるいは活用しようというアプローチである。

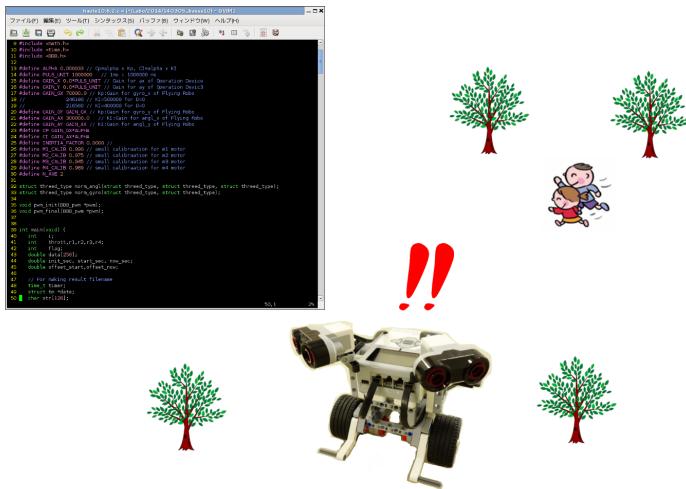


図 1.2: 知能（プログラム）が進化するロボット



図 1.3: 構成論的科学としてのロボット知能の進化

### ロボットの身体性

上でも述べたように、構成論的に知能を研究しようとすると、その「知能」を持った機械がある環境の中で行動することが必要である。いわば脳だけではだめで、身体があって「知能」を活用して行動することが必須である。

「知能」を搭載する機械のことを、筐体、そして「知能」と筐体の総

体のことを、ここではロボットと呼ぶことにしよう。ロボットという言葉の語感からは、アトムやアシモのような人間型の機械を想像してしまうかもしれない。本書では、広く「知能」を搭載した、行動（運動）することのできる機械（筐体）のことをロボットと呼ぶ。

自律運転の「知能」を搭載した自動車なども、単なる人間が操縦する機械ではなくて、もはやロボットと呼んでよいであろう。あるいは、自動飛行できるジャンボジェット機も、広い意味ではロボットと呼んだ方がよいかもしれない。

これらの例で言うと、自動車や飛行機の機械の部分のもつ性質のことを身体性と呼ぶ。また、筐体はロボットの「身体」と呼んでもよいであろう。

自動車と飛行機の例からも想像できるとおり、「知能」の構造や機能は身体性と大きく関わっている。あるいは、同じ「知能」であったとしても身体性の違いによって、生じる行動は大きく異なる可能性が高い。

## 創発

ロボットの行動は「知能」と「身体」だけで決定されるわけではない。どのような外部環境のなかでロボットが行動するかによって、生成される行動は異なる。いわば、「知能」、「身体」および環境の3者がダイナミックに相互作用するなかで、行動が現実に現れてくる。このことを創発とよぶ。

行動のための知能もそうであるが、様々なシステムはこの様に多くの構成要素からなっている。環境のなかでシステム全体が機能する時、それぞれの構成要素の機能の単なる和としてではなく、構成要素それ自身がもともと持っていないような機能が現れることを創発と言ふこともできる。個別分断的に構成要素を観察しても、全体としての機能を理解することはできない。

ロボットが置かれる環境は、人間が置かれている環境とは一般に異なる可能性がある。たとえば、深海に潜るロボットであったり、原子炉の

中に入るロボットであったり、空中を浮遊するロボットをイメージすれば、人間が普段置かれている環境条件とは異なる環境条件の中でロボットが行動することがわかる。

「知能」、「身体」および「環境」が相互作用することによって創発する行動がロボットインテリジェンスであるのだから、どのような環境条件の中で「知能」を実現するかということに依存して、結果として得られるロボット知能の形態は、人間のそれとは異なったものになることも十分考えられる。

### 飛行機の歴史から

飛行機の歴史を思い起こしてみる。鳥は、翼を利用して空を飛ぶ人間の先輩である。最初、人間は、翼が羽ばたくということに飛行のメカニズム原理が潜んでいると考えた。いまから振り返って考えると滑稽であるが、人工の翼を腕につけて多くの人々が飛行を試みたが、失敗に終わった。

その後、ライト兄弟につながる先人達は、鳥を徹底的に観察し、グライダーや飛行機を作って飛んだ。失敗を繰り返して飛行機の発明に至るその過程は、まさに飛行のメカニズムを解明するための構成論的科学であると言える。実際に飛行機を作って飛ばすという過程の中で、翼が空中で前進することによって揚力が発生するという飛行のメカニズムが解明されたのである。

現代のジャンボジェット機を見れば分かる通り、鳥を観察して構成された飛行機は、鳥とは異なる形をしている。しかし、推進力を生み出す方法が異なっているのであって、揚力を生み出す原理は両者に共通しているのである。

### ライト兄弟になれるのか

知能の研究の中で、まだわれわれはライト兄弟以前にいるということになる。知能ロボットを作って、実環境の中で行動させてみるということ

とを繰り返すことで前進しようとしている。先にも述べたとおり、このことを、構成論的科学と呼ぶ。一気に高度な知能を構成しようすることは、もちろん困難であろう。やはりもっとも基本的な、あるいは単純な知能から出発する。

そのために、ロボットの「知能」、すなわちロボット知能を大きく4つに分類して考えることからはじめる[?].

ひとつ目は、反応行動のための知能である。感覚器やセンサーから入力してきた情報にたいしてある定型的な反応を示す行動のことである。ここで、注意が必要である。もっとも単純であると考えられる反応行動においても、生成される（創発される）行動が定形的であるとは限らない。定型的なのは、反応を決めるメカニズム（感覚行動写像）である。反応のメカニズムが定型的であったとしても、複数の感覚入力が組み合わせられたり、環境との相互作用を通じて、複雑な行動が創発される可能性がある。

また、反応行動と混同しやすい言葉として、反射行動がある。反応行動の一種として反射行動が含まれる場合もあるが、反射行動とは、行動そのものが定形的な行動のことを指す。たとえば、熱いものに触ったときに、瞬間的に手を離す行動などである。

4つの知的行動の2つ目は、計画行動のための知能である。地図や経路情報をもとに行動の計画を構成する知能といえる。また、実際の行動から地図を構成する学習過程もこの知能に含めて考えられる。

3つ目が、適応行動のための知能である。変化する環境の下では、計画行動だけでは適応できない条件の変化に対応する能力が必要とされる。

4つ目として、協調行動のための知能がある。複数の知能を持った行動体が、それぞれの意図を理解し合いながら、目的を達成するための知能である。

これらの4つの知能は、それぞれの間に明確に線引きできるわけではなく、またそれが補いあったり、関連しあって全体として機能すると考えられる。ロボット知能を構成論的に考える上で、このように分けて考えることから出発しようとするわけである。本書では、これら4つ

の知能のなかで、主に反応行動のための知能に着目する。

### Debian をつかう

ロボットの「知能」は、具体的にはコンピューター内のプログラムの形をとる。なおかつ、環境のなかで実際に行動するロボットに搭載されるコンピューターなので、PCなどとことなり、小型かつ軽量であることが必要である。そのために、ボードコンピューターを利用する。ボードコンピューターにネットワーク接続してプログラムなどの調整を行う必要があるが、ロボットに搭載されているということもあり、PCのような入出力装置が存在しない。つまり、キーボードとディスプレイがない状態でプログラミングを行う必要がある。実際にはボードコンピューターにそれらを接続して利用することも可能であるが、構成論的に「知能」を構築する過程で、入出力装置をロボットに接続したり切り離したりという作業を常に行なうこととは、現実的ではない。

本書では、ボードコンピュータおよび基地となるPC上で利用するオペレーティングシステムとして Debian/GNU Linux（以下、単に Debian とよぶ）を用いる。Debian は GNU という名前がついていることからもわかるとおり、オープンソースとして管理されており、ロボット知能の研究のためには最適なOSのひとつとかんがえられる。

## 第2章 反応行動のための知能

ロボットは、図 2.1 に示したように、センサーを通じて環境からの情報を取り込む。その情報を基に行動を生成し、モーターを通して環境に働きかける。環境はロボットからの行動を受けて、そのダイナミクスにしたがって変化し、それを再びロボットがセンサーで受け取る。ロボットが環境の中で行動することは、このようなロボットと環境がつくるループを繰り返して行くことである。ロボットに着目すると、その構成要素

図 2.1: 環境と相互作用するロボット

はセンサー、感覚行動写像（知能）、モーターの 3 つに大別される。動物でいえば、それぞれ感覚器官、中枢神経系、筋肉に対応する。

センサーをロボットのどの部分に、どのような角度でつけるのか、あるいはモーターの数や大きさ角度をどのように形成するのか。これらの配置は、実際につくり出されるロボットの行動に大きく影響するであろう。このような要因は、**身体性**と呼ばれる。

一方、行動を決定しているのは**中枢神経系**であり、ロボットであればコンピュータでそれを決定することになる。反応行動を実現するメカニズムの一つとして、感覚行動写像がある[?]

## 2.1 走性における感覚行動写像

感覚入力は一つだけではなく、複数の値が考えられるのでベクトル  $\vec{s}$  でそれを表す。また、行動出力も複数ある場合を考えてベクトル  $\vec{r}$  で表す。従って、感覚行動写像は、感覚ベクトル  $\vec{s}$  から行動ベクトル  $\vec{r}$  への写像  $f$  となる。

$$f : \vec{s} \rightarrow \vec{r} \quad (2.1)$$

たとえば、図 2.2 に示したように、2つのセンサーと、2つのモーターを持つ走行ロボットを考えてみる。左側のセンサー値を  $s_L$ 、右側のセン

図 2.2: 2つのセンサーと2つのモーターをもつロボット例 (ncross)

サー値を  $s_R$  とする。また、左側のモーター出力を  $r_L$ 、右側のモーター出力を  $r_R$  とする。感覚ベクトル  $\vec{s}$  と、行動ベクトル  $\vec{r}$  は、それぞれ

$$\vec{s} = \begin{pmatrix} s_L \\ s_R \end{pmatrix}, \quad \vec{r} = \begin{pmatrix} r_L \\ r_R \end{pmatrix} \quad (2.2)$$

と表すことができる。

### 2.1.1 非交差性線形写像

図 2.2 に示した走行ロボットは、感覚入力と行動出力が左右で非交差性結合をしている (noncrossover link) ので、このロボットを ncross ロボットと呼ぶことにしよう。

センサーからの入力値  $s$  にゲイン  $g$  を掛けてそのまま出力値  $r$  とする感覚行動写像の場合,

$$\begin{aligned} r_L &= g_{LL} s_L \\ r_R &= g_{RR} s_R \end{aligned} \quad (2.3)$$

と書ける（線形写像）。ただし、左（L）のセンサーからの入力を左のモーターに出力する際のゲインを  $g_{LL}$  また、右（R）のセンサーからの入力を右のモーターに出力する際のゲインを  $g_{RR}$  と表した。行列  $\hat{G}_{\text{ncross}}$  を使ってこれを書くと、

$$\vec{r} = \hat{G}_{\text{ncross}} \vec{s} \quad (2.4)$$

$$\hat{G}_{\text{ncross}} \equiv \begin{pmatrix} g_{LL} & 0 \\ 0 & g_{RR} \end{pmatrix} \quad (2.5)$$

と、ゲイン行列  $\hat{G}_{\text{ncross}}$  は、 $2 \times 2$  の対角行列で表すことができる。

ゲインの値が正の場合、センサー値が大きくなればなるほど、モーター出力も大きくなるので、このことを興奮性結合とよぶ。すなわち、 $g_{LL} > 0, g_{RR} > 0$  のことを興奮性結合という。

このようなセンサーとモーターで構成される比較的単純な走行ロボットにおいてお、センサーの種類によって様々な走性が生じる。

## 光センサーによる走性

図 2.3 に示したように、センサーを光センサーとして、光源が左前方にある場合、走行ロボットはどのような振る舞いをするだろう？

左側のセンサーの値の方が大きいので、興奮性結合をもつロボットの場合、左のモーターの方が出力がより大きくなるであろう。結果として、光から遠ざかろうとする行動が生じると考えられる。

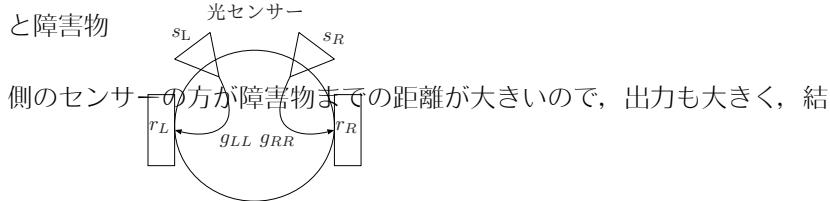
図 2.3: 光センサーをもつ走行ロボット (ncross-light) と光源

### 2.1.2 超音波センサーによる走性

センサーが、超音波センサー、すなわち距離センサーである場合にはどのような行動が生じるだろうか？(図 2.4 参照) 興奮性結合の場合、右



図 2.4: 超音波センサー（距離センサー）をもつ走行ロボット (ncross-ultra) と障害物



果として、障害物の方へぶつかっていく行動が生じると予想される。

### 2.1.3 交差性線形写像

図 2.5 に示した様に、センサーとモーターの結合が交差している場合を cross ロボットと呼ぶことにしよう。

図 2.5: 交差性結合をもつ走行ロボット (cross)

このロボットの場合、出力  $\vec{r}$  は

$$\vec{r} = \hat{G}_{\text{cross}} \vec{s} \quad (2.6)$$

$$\hat{G}_{\text{cross}} \equiv \begin{pmatrix} 0 & g_{RL} \\ g_{LR} & 0 \end{pmatrix} \quad (2.7)$$

と表される。

センサーが光センサーの場合、光がある方と逆のモーターの方が出力が大きくなるので、光の方に向かっていく走性を示すと予想される。

いっぽう、距離センサーを用いた場合、逆に障害物を避ける動きが予想される。

## 2.2 ゲインパラメータの取り得る範囲

ここまででは、簡単のためにゲインパラメータ  $\{g_{LL}, g_{RR}\}$  などの値については詳しく述べなかった。これらの値が正の場合、興奮性結合、すなわち

ち、センサー値が大きくなればなるほどモーター出力値も大きくなる場合を述べた。つまり、原理的にはモーター出力値はいくらでも大きな値を取りうる。

しかし、実際のロボットにおいてはモーターに対して無制限に大きな出力を動作させることは不可能である。センサー入力値の取り得る範囲を考慮して、モーター出力可能な範囲になるようにゲインパラメータの値を決める必要がある。

センサー値  $s$  の値が以下の範囲の値をとるとする。

$$s_{\min} \leq s \leq s_{\max} \quad (2.8)$$

線形写像の場合、センサー値にゲインを掛けたものをモーター出力値とするので、

$$\begin{aligned} gs_{\min} &\leq gs \leq gs_{\max} \\ gs_{\min} &\leq r \leq gs_{\max} \end{aligned} \quad (2.9)$$

である。

モーター出力値がとるべき範囲を

$$r_{\min} \leq r \leq r_{\max} \quad (2.10)$$

とすると、 $g$  の最大値に関して、

$$\begin{aligned} gs_{\max} &\leq r_{\max} \\ g &\leq \frac{r_{\max}}{s_{\max}} \end{aligned} \quad (2.11)$$

という条件を満たさなければならない。ここで、センサー値はセンサーからの光の強さや、障害物までの距離の値であるから、 $s_{\max} > 0$  と考えてよいことを用いた。

同様に、 $g$  の最小値について

$$g \geq \frac{r_{\min}}{s_{\min}} \quad (2.12)$$

でなければならない。

まとめると、ゲイン  $g$  は

$$\frac{r_{\min}}{s_{\min}} \leq g \leq \frac{r_{\max}}{s_{\max}} \quad (2.13)$$

の範囲内になければならない。

## 2.3 抑制性感覚運動写像

ここまででは、モーター出力値がセンサー入力値に比例して大きくなる線形写像を考えた。

ゲインの値( $-g$ )が負の値となる場合、センサー入力値が大きくなると、モーター出力値は小さくなる。モーター出力値が負になることが許される場合は、ここまで線形写像の表式をそのまま用いてもよいが、その値が負になることが出来ない場合には、定数  $c > 0$  を用いて

$$r = c - gs \quad (2.14)$$

とすることによってモーター出力値を正に保つことができる。この抑制性写像の場合にも、モーター出力値の取り得る範囲に応じて  $c$  と  $g$  の値の範囲を制限する必要があるのは、先に示した興奮性写像の場合と同様である。

## 2.4 非線形感覚運動写像

ここまででは、興奮性と抑制性の線形写像を用いる場合について述べた。非線形写像を用いることによって、より複雑な走性を創発する可能性がある。

たとえば、モーター出力値がセンサー値に反比例する以下のような場合、

$$r = \frac{g}{(s - s_{\min} + c)} \quad (2.15)$$

センサー値が大きくなればなるほどモーター出力が小さくなることは、抑制性線形写像と同じであるが、センサー値が小さい時にはモーター出力値が大きく変化し、センサー値が大きい場合にはモーター出力があまり変化しないという反応行動をつくることができる。

すなわち、この場合、

$$\frac{dr}{ds} = -\frac{g}{(s - s_{\min} + c)^2} \quad (2.16)$$

であるから、 $s_0 < s_1$  のとき

$$\left| \frac{dr}{ds}(s_0) \right| > \left| \frac{dr}{ds}(s_1) \right| \quad (2.17)$$

という性質をもつ感覚運動写像となる。

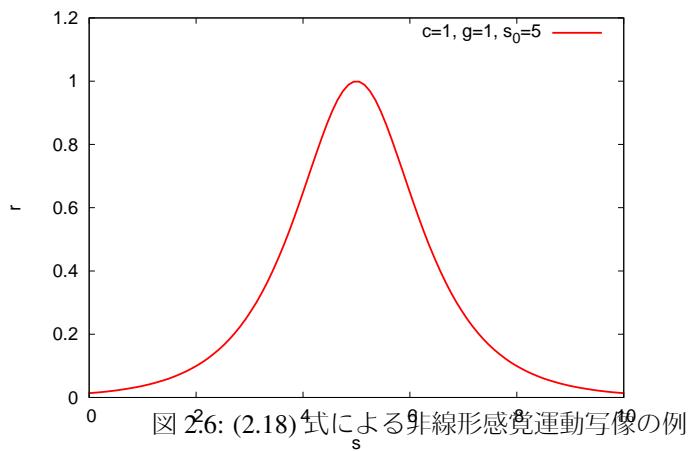
もうひとつの例として、興奮性および抑制性の2つの性質を同時にもつような写像の例をみてみよう。たとえば、

$$\begin{aligned} r &= \frac{c}{\cosh\{g(s - s_0)\}} \\ &= \frac{2c}{e^{g(s-s_0)} + e^{-g(s-s_0)}} \end{aligned} \quad (2.18)$$

の場合、 $s < s_0$  の領域では、興奮性であり、 $s > s_0$  の領域では抑制性の感覚運動写像となる（図2.6参照）。

## 2.5 感覚運動写像で行動するロボット

LEGO ロボットを使うと、ここまでに挙げた感覚行動写像で実際に走行するロボットを簡単につくることができる。第??章を参照してほしい。





## 第3章 収束する感覚行動写像

2014年ごろから、ドローンという言葉をニュースなどでも耳にするようになった。おもに、4つの回転翼をもつ飛行体（マルチコプター）のことをドローンと呼んでいるようである。単なるホビーとしてのラジコンから、空撮やインフラ調査撮影、また災害調査や農業への応用など爆発的な広がりを見せる勢いである。

このドローンは、単純な感覚運動写像で安定性を保っている飛行ロボットとみなすこともできる（図3.1参照）。



図3.1: 著者が開発した飛行ロボットの一例

ここでは、飛行ロボットの1軸回転運動をとりだし、単純な2階微分方程式でモデル化する[?]。感覚運動写像[?]に時間遅れやノイズが存在

しないと仮定した場合の、時間発展の振る舞いを解析する。

飛行ロボットの例として取り上げたが、この2階微分方程式は、摩擦のある振動や電気回路などをの挙動を表す方程式としてよく現れるものである。

解の求め方はいくつかある。変数分離や定数変化法を使って求める方法や、ラプラス変換して、それを部分分数にした後、逆変換して求める方法などが一般的かもしれない。ここでは、系の安定性を指数関数の指數部の値から直接議論しやすい行列形式を用いて解析する。

ここに示すように、系に時間遅れが存在しない場合、系は振動することはあっても、不安定になることは無いことを理論的に示す。ここでいう不安定とは振動の振幅が徐々に大きくなつて発散することを指す。

### 3.1 1軸回転運動

4つのローターを用いた飛行ロボットは、3次元空間を自由に並行移動し、回転の自由度も3つの軸に関してそれぞれ持っているので、合計6自由度の運動をする。

1軸(x-軸とする)の周りにおける回転運動のみを取り出して考えよう。図3.2に真横から飛行ロボットを見た図を示した。

また、図3.3に回転運動の概略図を示した。

水平状態からの回転角度 $\varphi$ の運動方程式は、慣性モーメント $I$ と回転軸の周りのトルク $I(L_1 - L_3)$ によって

$$I\ddot{\varphi} = I(L_1 - L_3) \quad (3.1)$$

と表される。ここで、 $L_1, L_3$ は1と3のそれぞれの位置における揚力である。また $I$ は回転軸から1および3の位置までの距離を表す。

いま感覚運動写像における感覚に当たるのは、ジャイロセンサーからの角速度 $\dot{\varphi}$ と、加速度センサーからの回転角度 $\varphi$ の2つであるとする。そして、運動に当たる出力は揚力 $L_1, L_3$ である。



図 3.2: 真横からみた飛行ロボット

図 3.3:  $x$ -軸周りの飛行ロボットの回転角度  $\varphi$  とローター 1,3 による揚力  $L_1, L_3$

線形写像を用いて感覚運動写像を

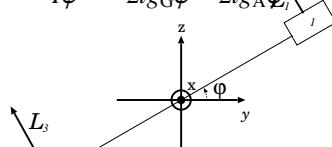
$$L_1 = -g_G \dot{\varphi} - g_A \varphi \quad (3.2)$$

$$L_3 = g_G \dot{\varphi} + g_A \varphi \quad (3.3)$$

とする。 $g_G > 0$  はジャイロセンサー値に対するゲイン、 $g_A > 0$  は加速度センサー値に対するゲインである。

揚力 (3.2),(3.3) を回転運動を表す (3.1) に代入すると、

$$I \ddot{\varphi} = -2l g_G \dot{\varphi} - 2l g_A \varphi \quad (3.4)$$



であるので、飛行ロボットの1軸回転運動は、

$$\ddot{\varphi} = -c_G \dot{\varphi} - c_A \varphi \quad (3.5)$$

と2階微分方程式で書ける。ただし、係数  $c_G, c_A$  を

$$c_G \equiv \frac{2Ig_G}{I} \quad (3.6)$$

$$c_A \equiv \frac{2Ig_A}{I} \quad (3.7)$$

と定義した。 $I > 0, I > 0$  であるから、これらの係数もそれぞれ  $c_G > 0, c_A > 0$  である。

時間  $t$  を明示的に示して書き直せば、

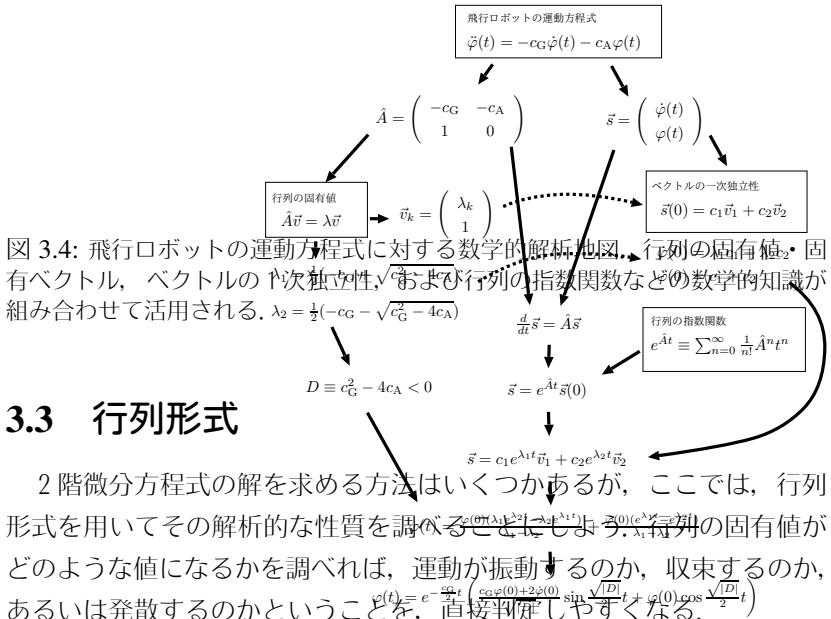
$$\ddot{\varphi}(t) = -c_G \dot{\varphi}(t) - c_A \varphi(t) \quad (3.8)$$

となる。この微分方程式は典型的な2階微分方程式であり、 $\varphi(t)$  の解析的な関数を明示的に求めることが可能である。つまり、飛行ロボットの1軸周りの運動はノイズや筐体の歪による影響を除いて、厳密に数学的にあらかじめ予測することが可能なはずである。

実際には、時間遅れがあるため、飛行ロボットの運動は(3.8)式で表される微分方程式の解とは異なった運動をするが、ここではひとまず、時間遅れがない場合、つまり(3.8)式の性質を明らかにする。

## 3.2 数学的解析の概略地図

飛行ロボットの運動を数学的に解析するためには、行列の固有値やベクトルの一次独立性、また関数の級数展開など、いくつか数学的知識が必要とする。これらの知識は、線形代数や解析学で個別に学ぶ項目であるが、それらの知見が単なるバラバラな存在としてだけでなく、それが関連して活躍し、飛行ロボットの運動という現実世界の問題を解析する上で結びつく。全体的な話の流れを図3.4に示した。



(3.8) 式から出発しよう。その左辺は、

$$\ddot{\varphi}(t) = \frac{d}{dt}\dot{\varphi}(t) \quad (3.9)$$

と書き改めることができる。また、すこしテクニカルな印象を受けるが、

次の自明な式

$$\frac{d}{dt}\varphi(t) = \dot{\varphi}(t) \quad (3.10)$$

を (3.8) 式と縦に並べて書くと,

$$\frac{d}{dt}\dot{\varphi}(t) = -c_G\dot{\varphi}(t) - c_A\varphi(t) \quad (3.11)$$

$$\frac{d}{dt}\varphi(t) = \dot{\varphi}(t) \quad (3.12)$$

となる。これらの微分方程式を行列形式で書くと

$$\frac{d}{dt} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} = \begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} \quad (3.13)$$

と書ける。さらに、角速度  $\dot{\varphi}(t)$  と角度  $\varphi(t)$  は、飛行ロボットの状態を表す量であるから、状態ベクトル  $\vec{s}(t)$  を

$$\vec{s}(t) \equiv \begin{pmatrix} \dot{\varphi}(t) \\ \varphi(t) \end{pmatrix} \quad (3.14)$$

と定義し、行列  $\hat{A}$  を

$$\hat{A} \equiv \begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \quad (3.15)$$

と定義すれば、(3.13) 式の行列形式は

$$\frac{d}{dt}\vec{s}(t) = \hat{A}\vec{s}(t) \quad (3.16)$$

と書きなおすことができる。つまり、(3.8) 式の 2 階微分方程式を行列  $\hat{A}$  を導入することによって、より簡潔な (3.16) 式の 1 階微分方程式の形に書き改めることができた。

### 3.4 行列の指數関数を用いた解

(3.16) 式は時間  $t$  に関する 1 階微分方程式のかたちをしている。その解  $\vec{s}(t)$  を具体的にもとめてみよう。 $\vec{s}(t)$  の  $n$  階微分を  $\vec{s}^{(n)}(t)$  と表す、つまり

$$\vec{s}^{(n)}(t) = \frac{d^n}{dt^n} \vec{s}(t) \quad (3.17)$$

と書き表すことにすると、テイラー級数を用いて、

$$\vec{s}(t) = \vec{s}(0) + \vec{s}^{(1)}(0)t + \frac{\vec{s}^{(2)}(0)}{2!}t^2 + \frac{\vec{s}^{(3)}(0)}{3!}t^3 + \dots \quad (3.18)$$

と書ける。"..."は、その先に無限に項がつづくことを表している。厳密には、この和が収束する条件を考慮する必要があるが、ここでは気にしないでこのように表せるとして、先に進む。 $\vec{s}(t)$ を1回微分するたびに、 $\hat{A}$ が掛けられるのであるから、 $n$ 回微分するということは、 $n$ 回 $\hat{A}$ を掛けることに等しい。

$$\begin{aligned} \vec{s}^{(n)}(t) &= \frac{d^n}{dt^n} \vec{s}(t) = \frac{d^{n-1}}{dt^{n-1}} \frac{d}{dt} \vec{s}(t) \\ &= \frac{d^{n-1}}{dt^{n-1}} \hat{A} \vec{s}(t) \\ &= \frac{d^{n-2}}{dt^{n-2}} \hat{A} \frac{d}{dt} \vec{s}(t) \\ &= \frac{d^{n-2}}{dt^{n-2}} \hat{A}^2 \vec{s}(t) \\ &\quad \dots \\ &= \hat{A}^n \vec{s}(t) \end{aligned} \quad (3.19)$$

ここで、行列 $\hat{A}$ は時間 $t$ に依存しないので、 $\hat{A}$ と、微分演算子 $\frac{d}{dt}$ が交換可能であることを使った。これを、テイラー展開の式に使うと、

$$\vec{s}(t) = \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!}t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!}t^3 + \dots \quad (3.20)$$

となることがわかる。つまり、(3.16)式が成り立つならば、(3.20)式が成り立つことがわかった。

逆に、(3.20)式が成り立つならば、(3.16)式が成り立つことを示そう。

(3.20) 式の両辺を  $t$  で微分すると,

$$\begin{aligned}\frac{d}{dt} \vec{s}(t) &= \frac{d}{dt} \left( \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!} t^3 + \dots \right) \\ &= \hat{A} \vec{s}(0) + \hat{A}^2 \vec{s}(0)t + \frac{\hat{A}^3 \vec{s}(0)}{2!} t^2 + \dots \\ &= \hat{A} \left( \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \dots \right) \\ &= \hat{A} \vec{s}(t)\end{aligned}\quad (3.21)$$

つまり, (3.16) 式

$$\frac{d}{dt} \vec{s}(t) = \hat{A} \vec{s}(t)$$

が成り立つ. その過程で, “...”の中に無限に項がたくさんあるという事実を使った. どんなにたくさん項があったとしても, その数が有限であるかぎり, このようなことは起こらない. 無限にあるからこそ, (3.16) 式が得られたのである.

ここでわかったことを, 改めて確認すると,

$$\frac{d}{dt} \vec{s}(t) = \hat{A} \vec{s}(t) \quad (3.22)$$

が成り立つならば,

$$\vec{s}(t) = \vec{s}(0) + \hat{A} \vec{s}(0)t + \frac{\hat{A}^2 \vec{s}(0)}{2!} t^2 + \frac{\hat{A}^3 \vec{s}(0)}{3!} t^3 + \dots \quad (3.23)$$

がその解として成り立つということである. また, その逆も成り立つことも確認した. つまり, 両者はお互いに必要十分条件になっている, すなわち等価であることがわかる.

上の,  $\vec{s}(t)$  の式をよく見ると,

$$\vec{s}(t) = \hat{f}(\hat{A}t) \vec{s}(0) \quad (3.24)$$

と書けることがわかる. ただし,

$$\hat{f}(\hat{A}t) \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!} t^2 + \frac{\hat{A}^3}{3!} t^3 + \dots \quad (3.25)$$

と定義した。ここで、 $\vec{v}(0)$  は状態ベクトルの初期値であるが、 $\hat{f}(\hat{A}t)$  という行列の右側に書かれていることに注意して欲しい。行列とベクトルの掛け算は、順序が入れ替わると意味が異なってくるのである。

このように、 $\vec{v}(t)$  に対する解は、形式的には簡単なかたちとして求めることができた。しかしそう見ると、こまつたことに、その行列の掛け算を無限回まで行い、それをすべて足し合わせなければならないことがわかる。いくらコンピューターの演算処理が高速化したといえども、無限回掛け算や足し算を繰り返すことはできない。

つぎに示す行列の固有値を用いると、この問題は解決される。

## 3.5 行列の固有値と固有ベクトル

いま、行列  $\hat{A}$  に対して、

$$\hat{A}\vec{v} = \lambda\vec{v} \quad (3.26)$$

を満たすべきベクトル  $\vec{v}$  とスカラー  $\lambda$  のことを、それぞれ  $\hat{A}$  の右固有ベクトル、および固有値という。これは、 $2 \times 2$  の次元をもつ行列  $\hat{A}$  についてのみ言えることではなく、一般の正方行列に対しても成り立つ。行列  $\hat{A}$  が対称行列であれば、右固有ベクトルと左固有ベクトルの区別はしなくても、両者は同じものになるが、(3.15) 式の行列の様に、非対称行列に関しては、右固有ベクトルと左固有ベクトルが異なるので、注意が必要である。ここではこのことに関して深入りはしない。ただし、後で具体的に求めるが、固有値、固有ベクトルは、行列の次数だけ存在するということは覚えておく必要がある。つまり、 $2 \times 2$  の行列の場合、固有値が 2 つ、そしてそれぞれの固有値に対応して、固有ベクトルも 2 つ存在する。

右固有ベクトルと固有値を求める方法については、後回しにして、ここでは一旦それらが得られたとして、話を進めることにする。(3.25) 式

$$\hat{f}(\hat{A}t) \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!}t^2 + \frac{\hat{A}^3}{3!}t^3 + \dots \quad (3.27)$$

の両辺に、右から  $\hat{A}$  の固有ベクトル  $\vec{v}$  を掛けると、

$$\begin{aligned}
 \hat{f}(\hat{A}t)\vec{v} &= \sum_{n=0}^{\infty} \frac{1}{n!} \hat{A}^n t^n \vec{v} \\
 &= \sum_{n=0}^{\infty} \frac{1}{n!} \hat{A}^n \vec{v} t^n \\
 &= \sum_{n=0}^{\infty} \frac{1}{n!} \lambda^n \vec{v} t^n \\
 &= \left( \sum_{n=0}^{\infty} \frac{1}{n!} \lambda^n t^n \right) \vec{v} \\
 &= e^{\lambda t} \vec{v}
 \end{aligned} \tag{3.28}$$

となる。固有値と固有ベクトルを使うと、行列をスカラーに置き換えられるので、行列の無限回掛け算という困難を指数関数  $e^{\lambda t}$  という形で、すっきり回避できるのである。

この式の意味するところは、行列  $\hat{f}(\hat{A}t)$  を  $\hat{A}$  の固有ベクトル  $\vec{v}$  に左から掛けることは、スカラー  $e^{\lambda t}$  を掛けることに等しいということである。そこで、

$$\hat{f}(\hat{A}t) = e^{\hat{A}t} \tag{3.29}$$

と書き表すことにすれば、

$$e^{\hat{A}t} \vec{v} = e^{\lambda t} \vec{v} \tag{3.30}$$

という、自然な形で表すことが出来る。つまり、行列  $\hat{A}$  の指数関数が、次のようにその無限級数で定義された。

$$e^{\hat{A}t} \equiv \hat{I} + \hat{A}t + \frac{\hat{A}^2}{2!} t^2 + \frac{\hat{A}^3}{3!} t^3 + \dots \tag{3.31}$$

さて、行列の固有値と右固有ベクトルはそれぞれ 2 つずつあるので、それらを  $\lambda_k, \vec{v}_k$  ( $k = 1, 2$ ) と表す。 $(3.28)$  式から、

$$e^{\hat{A}t} \vec{v}_k = \vec{v}_k e^{\lambda_k t} \quad (k = 1, 2) \tag{3.32}$$

である。また、右固有ベクトル  $\vec{v}_1, \vec{v}_2$  はそれぞれ、1 次独立であり、任意のベクトルをその線形結合で表すことができる。

## 3.6 状態ベクトルに対する解

準備が整ったので、状態ベクトル  $\vec{s}(t)$  の話に戻ろう。 (3.24) 式の  $\vec{s}(0)$  を右固有ベクトルの線形結合で表す。

$$\vec{s}(0) = c_1 \vec{v}_1 + c_2 \vec{v}_2 \quad (3.33)$$

これを (3.24) 式に代入すると、

$$\begin{aligned} \vec{s}(t) &= e^{\hat{A}t} (c_1 \vec{v}_1 + c_2 \vec{v}_2) \\ &= c_1 e^{\lambda_1 t} \vec{v}_1 + c_2 e^{\lambda_2 t} \vec{v}_2 \end{aligned} \quad (3.34)$$

となり、状態ベクトルは、行列の固有値によってその時間変化が記述されることがわかる。言い換えると、行列の固有値から、飛行ロボットの運動を記述（予測）できるということである。

ここで、行列  $\hat{A}$  の固有値と右固有ベクトルを求めておこう。行列の特性方程式<sup>1</sup>を用いれば、即座に固有値に対する方程式を求めることができるが、ここでは素直に固有値、固有ベクトルの定義から、それらを求めてみる。 (3.26) 式に具体的に行列 (3.15) を用いると、

$$\begin{pmatrix} -c_G & -c_A \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \lambda \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (3.35)$$

と具体的に書くことができる。ここで、右固有ベクトル  $\vec{v}$  を

$$\vec{v} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad (3.36)$$

と、 $\alpha, \beta$  を用いて表した。これらの値も、もちろん既知ではなく、固有値  $\lambda$  を求める過程において同時に求められるべきものである。 $2 \times 2$  行列とベクトルの積を展開してみると、

$$-c_G\alpha - c_A\beta = \lambda\alpha \quad (3.37)$$

$$\alpha = \lambda\beta \quad (3.38)$$

---

<sup>1</sup> $\det(\hat{A} - \lambda\hat{I}) = 0$  を行列  $\hat{A}$  に対する特性方程式よ呼び、これを満たす  $\lambda$  を求めれば、それが固有値となる。ただし、 $\hat{I}$  は単位行列。 $\det$  は行列式をとることを意味する。

となる。 (3.38) 式を (3.37) 式に代入すると,

$$-c_G \lambda \beta - c_A \beta = \lambda^2 \beta \quad (3.39)$$

である。任意の  $\beta$  に対してこれが成り立つためには

$$\lambda^2 + c_G \lambda + c_A = 0 \quad (3.40)$$

である必要があり、2次方程式の解の公式を用いて固有値  $\lambda$  は

$$\lambda_1 = \frac{1}{2} \left( -c_G + \sqrt{c_G^2 - 4c_A} \right) \quad (3.41)$$

$$\lambda_2 = \frac{1}{2} \left( -c_G - \sqrt{c_G^2 - 4c_A} \right) \quad (3.42)$$

と求められる。規格化条件  $|t| = 1$  がない場合には、 $\alpha, \beta$  の値は一意には定まらないが、その比は (3.38) 式によって与えられる。

### 3.7 状態ベクトルの具体的なたたち

ここまで議論では、(3.41),(3.42) 式であったえられる固有値  $\lambda_k$  が複素数になる可能性がある。同様に (3.38) 式から、右固有ベクトルも複素ベクトルになる可能性がある。したがって、(3.34) 式で与えられる状態ベクトルも複素ベクトルになる可能性がある。状態ベクトルの要素は、もともと飛行ロボットの姿勢角度と角速度であるから、実数でなければならない。そもそも初期状態  $s(0)$  が実数ベクトルであることからしても、奇妙に思える。そこで、初期状態を用いて状態ベクトルを具体的に表してみよう。初期状態  $t = 0$  においては、

$$\begin{pmatrix} \dot{\varphi}(0) \\ \varphi(0) \end{pmatrix} = c_1 \begin{pmatrix} \lambda_1 \\ 1 \end{pmatrix} + c_2 \begin{pmatrix} \lambda_2 \\ 1 \end{pmatrix} \quad (3.43)$$

であるので、この連立方程式から  $\lambda_1 \neq \lambda_2$  の場合の  $c_1, c_2$  を求めると、

$$c_1 = \frac{\lambda_2 \varphi(0) - \dot{\varphi}(0)}{\lambda_2 - \lambda_1} \quad (3.44)$$

$$c_2 = -\frac{\lambda_1 \varphi(0) - \dot{\varphi}(0)}{\lambda_2 - \lambda_1} \quad (3.45)$$

となる。したがって、これらを(3.34)式

$$\vec{s}(t) = c_1 e^{\lambda_1 t} \vec{v}_1 + c_2 e^{\lambda_2 t} \vec{v}_2$$

に用いると、

$$\varphi(t) = \varphi(0) \left\{ \frac{\lambda_2 e^{\lambda_1 t} + \lambda_1 e^{\lambda_2 t}}{\lambda_1 - \lambda_2} \right\} + \dot{\varphi}(0) \left\{ \frac{e^{\lambda_1 t} - e^{\lambda_2 t}}{\lambda_1 - \lambda_2} \right\} \quad (3.46)$$

と得られる。無論、角速度  $\dot{\varphi}(t)$  も同様にして求めることが可能であるし、この式を時間微分して求めることもできる。

(3.46)式には、行列  $\hat{A}$  の固有値  $\lambda_1, \lambda_2$  が、まだそのまま含まれている。その値がどのような値をとっても、一般的に成り立つ式であると言える。しかし、その値が複素数である場合には、実関数であるべき  $\varphi(t)$  が本当に実数値を取り得る関数であるかどうか分かりにくい。

そこで、固有値を与える(3.41),(3.42)式

$$\begin{aligned} \lambda_1 &= \frac{1}{2} \left( -c_G + \sqrt{c_G^2 - 4c_A} \right) \\ \lambda_2 &= \frac{1}{2} \left( -c_G - \sqrt{c_G^2 - 4c_A} \right) \end{aligned}$$

を具体的に用いてみよう。固有値の式からわかる通り  $c_G^2 - 4c_A$  の値が正の時には、固有値の虚数部はゼロである。したがって、当然  $\varphi(t)$  の値も実数値となる。いっぽうその値が負の時には、固有値は複素数になる。そこで、

$$D \equiv c_G^2 - 4c_A \quad (3.47)$$

と定義して、 $\varphi(t)$  の式を具体的に表してみると、

$$\begin{aligned} \varphi(t) &= \frac{\varphi(0)}{2} e^{-\frac{c_G}{2} t} \left\{ \frac{c_G}{\sqrt{D}} \left( e^{\frac{\sqrt{D}}{2} t} - e^{-\frac{\sqrt{D}}{2} t} \right) + \left( e^{\frac{\sqrt{D}}{2} t} + e^{-\frac{\sqrt{D}}{2} t} \right) \right\} \\ &\quad + \frac{\dot{\varphi}(0)}{\sqrt{D}} e^{-\frac{c_G}{2} t} \left( e^{\frac{\sqrt{D}}{2} t} - e^{-\frac{\sqrt{D}}{2} t} \right) \end{aligned} \quad (3.48)$$

となる。 $D < 0$  のときに虚数が現れる。

$$\sqrt{D} = i \sqrt{|D|} \quad (3.49)$$

ここで、 $i$  は虚数単位である。これを用いると、

$$\begin{aligned}\varphi(t) &= \frac{\varphi(0)}{2} e^{-\frac{c_G}{2}t} \left\{ \frac{c_G}{i\sqrt{|D|}} \left( e^{\frac{i\sqrt{|D|}}{2}t} - e^{-\frac{i\sqrt{|D|}}{2}t} \right) + \left( e^{\frac{i\sqrt{|D|}}{2}t} + e^{-\frac{i\sqrt{|D|}}{2}t} \right) \right\} \\ &\quad + \frac{\dot{\varphi}(0)}{i\sqrt{|D|}} e^{-\frac{c_G}{2}t} \left( e^{i\frac{\sqrt{|D|}}{2}t} - e^{-i\frac{\sqrt{|D|}}{2}t} \right) \\ &= \varphi(0) e^{-\frac{c_G}{2}t} \left\{ \frac{c_G}{\sqrt{|D|}} \sin \frac{\sqrt{|D|}}{2}t + \cos \frac{\sqrt{|D|}}{2}t \right\} \\ &\quad + \frac{2\dot{\varphi}(0)}{\sqrt{|D|}} e^{-\frac{c_G}{2}t} \sin \frac{\sqrt{|D|}}{2}t\end{aligned}\tag{3.50}$$

となり、すべて実数と実数の三角関数を用いて表されるので、確かに  $\varphi(t)$  は実数関数であることが確認できる。

この表式の物語っていることはどのようなことであろうか？大きき分けると、初期角度  $\varphi(0)$  に関する項と、初期角速度  $\dot{\varphi}(0)$  に関する項の 2 つの項から全体が構成されていることがわかる。

もうすこし  $\varphi(t)$  の式の意味するところを見やすくするためにここで、減衰係数  $\gamma$  を

$$\gamma \equiv \frac{c_G}{2}\tag{3.51}$$

と定義し、角振動数  $\omega$  を

$$\omega \equiv \frac{\sqrt{|D|}}{2}\tag{3.52}$$

と、それぞれ定義すると、

$$\varphi(t) = \varphi(0) e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t\tag{3.53}$$

といふかたちに整理できる。ただし、ここまで話しあくまでも  $D < 0$  の場合にたいするものである。

$c_G, c_A$  は、運動方程式においては、その意味がわかり易かった。しかし、この時間発展の式のなかでそれを直接使うと、煩雑なかたちとなる。むしろ、 $\gamma, \omega$  を用いて表した方が、よりその振る舞いがイメージしやすくなる。

### 3.8 無矛盾性の確認

ここまで、 $D = c_G^2 - 4c_A < 0$  の条件が満たされる場合について  $\varphi(t)$  の具体的な表式をもとめた。どうやって、最終結果  $\varphi(t)$  に間違いがないことを確認すればよいだろう？

(3.53) 式が、実際に (3.8) 式

$$\ddot{\varphi}(t) = -c_G \dot{\varphi}(t) - c_A \varphi(t)$$

を満たすことを確認する。 $\varphi(t)$  を時間微分して、 $\varphi(0), \dot{\varphi}(0)$  および三角関数の種類別にまとめると、

$$\dot{\varphi}(t) = -\left(\frac{\gamma^2}{\omega} + \omega\right)\varphi(0)e^{-\gamma t} \sin \omega t \quad (3.54)$$

$$-\frac{\gamma}{\omega}\dot{\varphi}(0)e^{-\gamma t} \sin \omega t \quad (3.55)$$

$$+\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \quad (3.56)$$

となる。さらに時間微分して、 $\ddot{\varphi}(t)$  を求めると、

$$\begin{aligned} \ddot{\varphi}(t) &= \frac{\gamma}{\omega}(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \sin \omega t \\ &\quad -(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \cos \omega t \\ &\quad +\left(\frac{\gamma^2}{\omega} - \omega\right)\dot{\varphi}(0)e^{-\gamma t} \sin \omega t \\ &\quad -2\gamma\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \end{aligned} \quad (3.57)$$

と求められる。これが、(3.8) 式の左辺である。ここで、

$$\gamma = \frac{c_G}{2}, \quad \omega = \frac{\sqrt{|D|}}{2}, \quad D = c_G^2 - 4c_A < 0$$

を使うと、

$$\gamma^2 + \omega^2 = c_A \quad (3.58)$$

という関係が成り立つ。

さらに、これらの関係を使うと、(3.8)式の右辺は、

$$-c_G\dot{\varphi}(t) - c_A\varphi(t) = -2\gamma\dot{\varphi}(t) - (\gamma^2 + \omega^2)\varphi(t) \quad (3.59)$$

と表される。ここに  $\dot{\varphi}(t)$  および  $\varphi(t)$  の各項を代入すればよい。すこし煩雑になるので、右辺について、各項べつにまとめると、

$$1. \varphi(0)e^{-\gamma t} \sin \omega t \text{ の係数 : } \frac{\gamma}{\omega}(\gamma^2 + \omega^2)$$

$$2. \varphi(0)e^{-\gamma t} \cos \omega t \text{ の係数 : } -(\gamma^2 + \omega^2)$$

$$3. \dot{\varphi}(0)e^{-\gamma t} \sin \omega t \text{ の係数 : } \frac{\gamma^2}{\omega} - \omega$$

$$4. \dot{\varphi}(0)e^{-\gamma t} \cos \omega t \text{ の係数 : } -2\gamma$$

となり、たしかに  $\ddot{\varphi}(t)$  の各項の係数

$$\begin{aligned} \ddot{\varphi}(t) &= \frac{\gamma}{\omega}(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \sin \omega t \\ &\quad -(\gamma^2 + \omega^2)\varphi(0)e^{-\gamma t} \cos \omega t \\ &\quad + \left( \frac{\gamma^2}{\omega} - \omega \right) \dot{\varphi}(0)e^{-\gamma t} \sin \omega t \\ &\quad - 2\gamma\dot{\varphi}(0)e^{-\gamma t} \cos \omega t \end{aligned}$$

と一致する。

$\varphi(t)$  の具体的表式は、その出発点にした2階微分方程式をたしかに満たしていることが確認できた。行列形式や固有ベクトルなどの道具をつかって、たどり着いた目的地がもとの出発点と矛盾していないことが分かった。

数学的な論理展開に象徴されるような、「論理」というものは、意外と脆い側面がある。それぞれの論理には、それが成り立つ条件というものがある。そのような条件付き論理を、いくつも重ねて用いてたどり着く結論というものには、当然それぞれの条件が何重にも課されているので、ただでさえ危うい。さらに加えて、無矛盾性が確認されていない論理展開というものは、そうやすやすと信じてよいとは言えないものである。

### 3.9 振動

はなしを,  $\varphi(t)$  すなわち, 飛行ロボットの傾き角度の具体的な振るまいにもどそう.  $D = c_G^2 - 4c_A < 0$  の場合には,  $\varphi(t)$  は(3.53)式

$$\varphi(t) = \varphi(0)e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t \quad (3.60)$$

と表される.

$c_G, c_A$  は, われわれが選べるパラメーターであった. いま,  $c_G = 0$  と選ぶと,  $\gamma = c_G/2 = 0$  であるから,

$$\varphi(t) = \varphi(0) \cos \omega t + \frac{\dot{\varphi}(0)}{\omega} \sin \omega t \quad (3.61)$$

である. また, これを時間微分すると

$$\dot{\varphi}(t) = -\varphi(0)\omega \sin \omega t + \dot{\varphi}(0) \cos \omega t \quad (3.62)$$

が得られる.

これは, 式からも分かるとおり, 振幅が初期状態  $\varphi(0), \dot{\varphi}(0)$  に依存する時間的な振動運動を表す. 角振動数  $\omega$  は, いま  $c_G = 0$  であるから,  $\omega = \sqrt{|D|}/2 = \sqrt{c_A}$  と,  $c_A$  の値だけで決まる. ただし,  $D < 0$  でなければならぬので,  $c_A > 0$  の条件を満たさなければならない.

$\omega = \pi, 3/2\pi, 1/2\pi$  の場合の,  $\varphi(t), \dot{\varphi}(t)$  の時間変化を図 3.5 に表した.  $\omega$  の値が大きいほど振動の周期が短い.  $\omega$  の値は,  $c_A$  の値だけできまるのであるから,  $c_A$  の値, すなわち, PI 制御の比例ゲインはロボット運動の振動周期を左右するパラメーターであることがわかる.

$\varphi(t)$  と  $\dot{\varphi}(t)$  はおたがいに位相がすこしづれた, 似たような振動を続けることがわかった.

ここで, つぎの量を計算してみると,

$$\{\omega\varphi(t)\}^2 + \dot{\varphi}(t)^2 = \omega^2\varphi(0)^2 + \dot{\varphi}(0)^2 \quad (3.63)$$

と, 右辺は時間  $t$  に依存しない一定値となることがわかる. つまり,  $\omega\varphi(t)$  を横軸,  $\dot{\varphi}(t)$  を縦軸としてグラフを描くと, 半径が  $\sqrt{\omega^2\varphi(0)^2 + \dot{\varphi}(0)^2}$  の円となることを意味している.

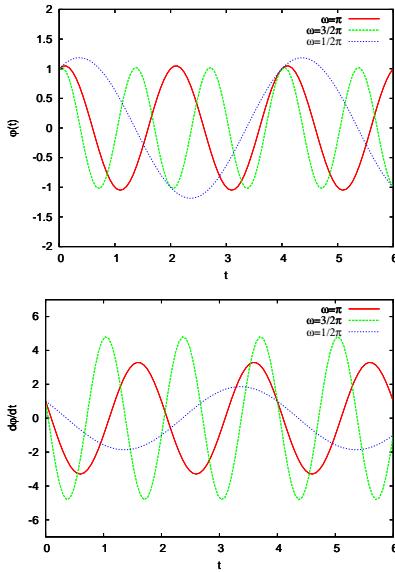


図 3.5:  $c_G = 0$  の場合,  $\varphi(t), \dot{\varphi}(t)$  の時間変化は振動をつづける.

実際に, それらを図 3.6 に描画した. 確かに, 円が描かれた.

この空間のことを, **相空間**と呼び, 運動状態はこの空間内の 1 点として表される. また, 運動の時間発展は, 相空間内の軌道として描かれることがある.  $c_G = 0$  の場合の振動運動は, 円軌道として表される.

### 3.10 減衰振動

具体的な時間発展の式

$$\varphi(t) = \varphi(0)e^{-\gamma t} \left( \frac{\gamma}{\omega} \sin \omega t + \cos \omega t \right) + \frac{\dot{\varphi}(0)}{\omega} e^{-\gamma t} \sin \omega t$$

が意味するところを理解するために, 各項についてそれぞれくわしく見てみる.

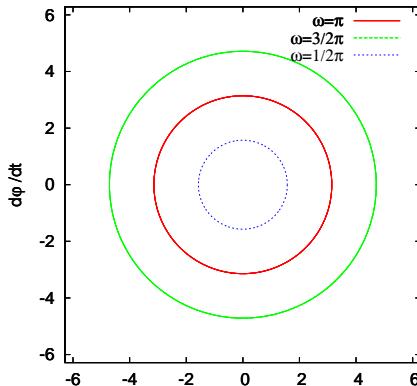


図 3.6:  $\omega\varphi(t), \dot{\varphi}(t)$  をそれぞれ横軸と縦軸に表した空間を相空間と呼ぶ。運動は、相空間の軌道として表される。振動運動は、相空間における円軌道となる。

第1項目は初期角度  $\varphi(0)$  がゼロでない時に現れる項である。その全体に  $e^{-\gamma t}$  が掛かっているので、 $\gamma > 0$  すなわち  $c_G > 0$  の場合には、初期角度の効果は時間が経てば減衰していくことを意味している。そして、減衰係数が大きければ大きいほど、速く減衰する（図 3.7(a) 参照）。次に、こ

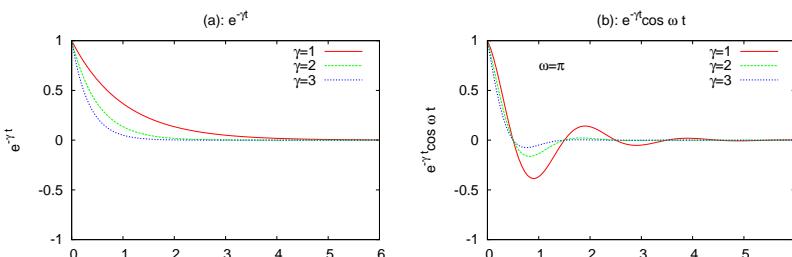


図 3.7: (a) 指数関数的な減衰の例。減衰係数  $\gamma$  が大きいほど、速く減衰する。  
(b) 振幅が指数関数的に減衰する  $\cos$  関数の例。

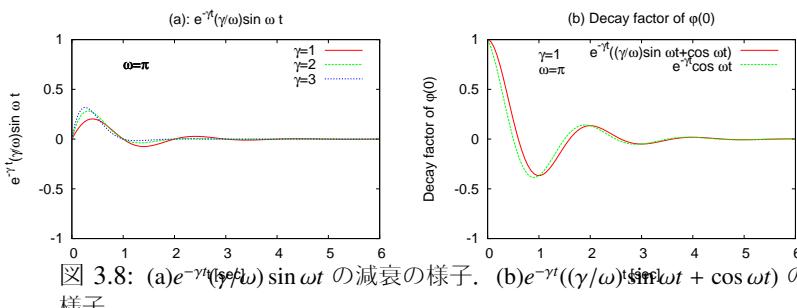
の初期角度の減衰項の中の、 $\cos$  関数で表された項がどのように時間変化

するのかを図3.7(b)に示した。例として角振動数  $\omega = \pi$  の場合を示した。振動しながら、その振幅が減衰していくことがわかる。このような関数の振る舞いの場合に、 $e^{-\gamma t}$  のことを包絡関数、あるいは包絡線と呼ぶ。

$\varphi(t)$  の時間発展を表す式の、第1項にはもうひとつの項が含まれていることを忘れてはならない。この項は

$$e^{-\gamma t} \frac{\gamma}{\omega} \sin \omega t \quad (3.64)$$

というかたちをしている。この項自身の減衰の様子を図3.8(a)に示した。この項が  $\cos$  関数の減衰に加えられたものが実際の振動減衰の時間発展



(図3.8(b)参照)である。 $\cos$  関数だけの場合と比べて、やや位相が遅れた減衰を示すが、結局指数関数的に減衰する包絡線によってその振動は抑えられて、急速にゼロに収束する様子がわかる。

さらに初期角速度の項があるが、この項は初期角度の  $\sin$  関数と類似のかたちをしており、その時間発展の様子も、まったく同様の振る舞いをすることが容易に理解できる。

### 3.11 比例ゲインと積分ゲイン空間における相図

次に、固有値を与える(3.41))式と、 $c_G, c_A$  の値を用いて、 $\lambda_1$  が実際にどのような値をとるかを図示してみよう。それらの実数部の正負によっ

て、状態ベクトルの発散、収束が決まる。図 3.9 に  $\lambda_1$  の実部と虚部の値

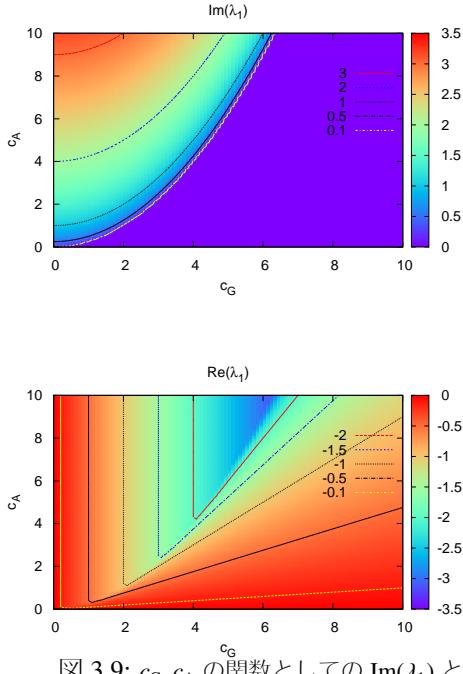


図 3.9:  $c_G, c_A$  の関数としての  $\text{Im}(\lambda_1)$  と  $\text{Re}(\lambda_1)$

を  $c_G, c_A$  の関数として示した。まず、虚部を見ると、 $c_A < c_G^2/4$  の領域では、 $\text{Im}(\lambda_1)=0$  なので、系は振動しない。一方  $c_A > c_G^2/4$  の領域では、 $\text{Im}(\lambda_1) > 0$  となり、系は振動する。 $\text{Im}(\lambda_1)$  の大きさがその振動数に対応するが、 $c_A = c_G^2/4$  から離れれば離れるほど、振動数が大きくなることがわかる。

つぎに、実数部を見るとすべての領域で負の値になる。包絡曲線は図に示したすべての領域でゼロに収束することがわかる。境界  $c_A = c_G^2/4$  の近傍において、比較的  $|\text{Re}(\lambda)|$  のあたいが大きくなる、すなわち速く収

束する傾向にあることがわかる。

### 3.12 遷移行列を用いた漸化式

このように、系の振る舞いは行列の固有値を求める問題に帰着され、その固有値がもともとの系のゲインの値  $c_G, c_A$  によって決まることがわかる。すでに明示的に系の振る舞いが表現されているが、ここでは漸化式の形式を用いても以上の議論を定式化可能であることを示す。

微分方程式 (3.16) から、微小時間  $\Delta t$  に対して

$$\Delta \vec{s}(t) = \hat{A} \vec{s}(t) \Delta t \quad (3.65)$$

が成り立つ。これを用いると  $\Delta t$  後、すなわち  $t + \Delta t$  における状態ベクトル  $\vec{s}(t + \Delta t)$  は

$$\begin{aligned} \vec{s}(t + \Delta t) &= \vec{s}(t) + \Delta \vec{s}(t) \\ &= \vec{s}(t) + \hat{A} \vec{s}(t) \Delta t \\ &= (\hat{I} + \Delta t \hat{A}) \vec{s}(t) \end{aligned} \quad (3.66)$$

となる。ここで、 $\hat{I}$  は単位行列

$$\hat{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (3.67)$$

である。また、スカラー量  $\Delta t$  と行列やベクトルとの積は、順序に依存しない性質を用いた。 $\vec{s}_{i+1} = \vec{s}(t + \Delta t)$ ,  $\vec{s}_i = \vec{s}(t)$  と置き換えて書くと (3.66) 式は

$$\vec{s}_{i+1} = \hat{T} \vec{s}_i \quad (3.68)$$

と漸化式のかたちに書くことが出来る。ただし、ここで遷移行列  $\hat{T}$  を

$$\hat{T} \equiv \hat{I} + \Delta t \hat{A} \quad (3.69)$$

と定義した。行列  $\hat{A}$  の右固有ベクトル  $\vec{v}$  を用いると

$$\begin{aligned}\hat{T}\vec{v} &= (\hat{I} + \Delta t \hat{A})\vec{v} \\ &= (1 + \Delta t \lambda)\vec{v}\end{aligned}\quad (3.70)$$

であるから、遷移行列  $\hat{T}$  の固有値  $\eta$  は

$$\eta = 1 + \Delta t \lambda \quad (3.71)$$

である。したがって、元の  $\hat{A}$  の固有値  $\lambda$  は、遷移行列の固有値  $\eta$  を用いて

$$\lambda = \frac{\eta - 1}{\Delta t} \quad (3.72)$$

と求められる。

この関係を用いれば、漸化式 (3.68) から、系の振る舞いを決める  $\hat{A}$  行列の固有値  $\lambda$  を求めることができる。



# 第4章 漸化式と数値解析

数学は、様々な量を文字記号として象徴的に表し、数式としてそれらの間の関係性を客観的に表現する方法である。人類が2千年もの年月のあいだ使ってきたものである。しかし、万能ではない。数学の力で（解析的に）表現したり答えを求めたりできない場合も多々ある。そのような場合には、コンピュータ上で数値解析を行うのが、一つの強力な解決法である。

つまり、数値解析とは、端的に言えば、解析的に解くことが困難な問題を、コンピュータを用いて数値的に解決する方法である。

ここでは、はじめに数値解析を含めてプログラミングにおいて重要な概念である構造化プログラミングを基に、正しく機能するプログラムをつくるために注意すべき点について述べる。ここで「正しく」と言うのは、「求められている機能を常に果たすことができる」という意味である。具体的な例としてニュートン・ラフソン法について述べる。

また、数値解析を行う上で重要な、打ち切り誤差や丸め誤差などの計算誤差についても述べる。

## 4.1 正しく機能するプログラムを書く

### データの流れを明瞭にする

アルゴリズムの具体的なコンピュータ上の実態をプログラムと言う。また、そのプログラムを作成することをプログラミングという。

プログラミングにおいて必須なことは正しく機能するプログラムをつくるということである。より高速に動作することや、技巧的なアルゴリ

ズムを用いて効率を上げることも時には必要であるが、正しさを損なう危険性がある場合には、むしろ単純さを追求することが重要である。

このことは、プログラミング言語の種類に関係なく必要なことである。オブジェクト指向言語であろうが、そうでなかろうが、あるいは、手続き型であろうがイベントドリブン型であろうがなかろうが、Java言語であろうがC言語であろうが、正しく機能するプログラムを求めるることは、あらゆるプログラミングの場面において普遍的に必要なことである。

正しいプログラムをつくるためには、構造化プログラミングの考え方則ってそれを行う必要がある。

## 4.2 構造化プログラミング

正しく動作するプログラムとは、言い換えると、常に必要とする機能を果たして結果を返してくれるということである。以下では、一例として実数の平方根を求めるプログラムを考えるが、データによってはエラーを起こしたり、間違った答えを返すプログラムは、正しいとはいえない。

### データの流れを明瞭にする

アルゴリズムとは、言い換えれば、データの流れのことである。プログラムを正しく動作するようにつくるためには、データの流れがわかりやすく記述されている必要がある。データの流れがわかりやすく明瞭であるために、技巧的になりすぎず、単純化されたプログラムをつくる必要がある。つまり、うまいプログラムは必要ない。わかりやすく書こう。

### 基本アルゴリズムを組み合わせる

わかりやすく単純化されたプログラムをつくるといっても、プログラムの果たすべき機能が複雑な場合にはどうすればいいのだろうか？

複雑なアルゴリズムを、より単純な機能の組み合わせとして記述するのである。もっとも基本的なアルゴリズムとして以下の3つの機能がある。

1. 命令（演算や関数）
2. 繰り返し（While, for, …）
3. 条件分岐（if, case, switch, …）

以上の3つの基本アルゴリズムを組み合わせることによって、あらゆる複雑なアルゴリズムを構成することが可能であるというのが、構造化プログラミング [?, ?] の考え方である。

### goto 文をつかわない

正しいプログラムをつくるためには、データの流れを明瞭にする必要がある。goto 文をつかうと、プログラムの任意の箇所から、任意の箇所へ処理を飛ばすことができるが、これを使うと、データの流れが不明瞭になる場合が多い。

### 関数をつかう

単純化のもうひとつ重要な要素として、main ルーチン (main 関数) を長く複雑にしがちで済むようになることが理想的である。

そのためには、関数あるいはサブルーチンを使う。プログラミングにおいて、それらの役割は、一連のひとまとまりの処理を定義して、その処理を行いたい箇所で、その関数（あるいはサブルーチン）を呼び出すだけで済むようにすることである。

関数とサブルーチンの違いは、引数を渡して値を返すかたちに定義するか、あるいは単なる処理の集まりとして定義するかの違いだけである。

関数とは、一連の処理が、ひとつの関数名として定義されたものであるから、関数の役割は、抽象化を実現することと言う事も出来る。つまり

り、関数名が抽象化されたシンボルであり、その具体的な内容がその関数の定義部分の一連の処理ということである。

### 4.3 ニュートン・ラフソン法

漸化式を繰り返し適用することによって、関数のゼロ切片を求める方法の一例として、ニュートン・ラフソン法を説明する。単にニュートン法と呼ばれる場合もある。

変数  $x$  によって微分可能な関数  $f(x), g(x)$  が

$$y = f(x) = g(x) - C \quad (4.1)$$

という関係にあるとき、 $f(x) = 0$  を満たす  $x$  を求めれば、 $g(x) = C$  を満たす  $x$  の値を求めることができる。ここで、 $C$  は定数である。図4.1に示

図4.1: ニュートン・ラフソン法の原理

したように、 $x_0$  における  $f(x)$  の接線が  $x$ -軸と交わる点を  $x_1$  とすると、

微分係数の意味から、

$$\frac{df}{dx}(x_0) = \frac{f(x_0)}{x_0 - x_1} \quad (4.2)$$

である。したがって、 $x_1$  は

$$x_1 = x_0 - \frac{f(x_0)}{\frac{df}{dx}(x_0)} \quad (4.3)$$

と求められる。同様に、 $x_n$  の値から  $x_{n+1}$  の値が

$$x_{n+1} = x_n - \frac{f(x_n)}{\frac{df}{dx}(x_n)} \quad (4.4)$$

という漸化式を用いて求めることができる<sup>1</sup>。これを繰り返えすと  $x_n$  は  $g(x) = C$  を満たす  $x$  の値に近づく。

この様に、同様の計算手続きを繰り返すことによって、目的の値に近づいていくアルゴリズムのことを反復法という。

## $\sqrt{2}$ を求める具体例

具体的に、 $g(x) = x^2$  また、 $C = 2$  とすると  $x^2 = 2$  を満たす  $x$  の値、すなわち  $\sqrt{2}$  の値を求めることになる。図 4.2 にそのプログラム例を示した。この例では、基本アルゴリズム要素のなかの、命令と繰り返しを用いている。その意味では、構造化プログラムと言える。

## インデントと数学的表式に近い変数名

しかし、次に示すように、適切なインデント（字下げ）と、数学的表式と混乱を招かない変数名を用いた方が、よりデータの流れが明確になる。また、図 4.3 の例では、 $C = 2$  以外の場合にも  $\sqrt{C}$  の値を求めることができるように、一般化されている。繰り返しの終了判定を while(条件)

---

<sup>1</sup> 数列などの、項と項の間に一定の関係があり、初項からはじまって、つぎつぎと項を求めることが出来る式を漸化式という。たとえば、等比数列の場合、一つ前の項に等比を掛けたものが次の項となる。

```

1 #include <stdio.h>
2 #include <math.h>
3 int main(){
4     double a,b,c,x;
5     x=2.0;
6     a=x;
7     b=(x/a+a)/2.0;
8     c=b-a;
9     while(fabs(c)>1.0e-6){
10         c=b-a;
11         printf("%12.10f \n",b);
12         a=b;
13         b=(x/a+a)/2.0;
14     }
15 }
```

図 4.2: ニュートン・ラフソン法で  $\sqrt{2}$  の値を求める。適切にインデントされておらず、変数名も混乱を招く。

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main(){
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    x=C;
16    dx=- (x*x-C) / (2.0*x);
17    while(fabs(dx)>eps){
18        x=x+dx;
19        printf("%12.10f \n",x);
20        dx=- (x*x-C) / (2.0*x);
21    }
22 }
23 }
```

図 4.3: 適切なインデントと数学的表式を連想できる変数名を用いたプログラム。

件) で行なっているが、そこに現れる微小量を条件の中に直接記述するのではなく、`eps` として定数定義している（3 行目）。こうすることによっ

て、プログラムの内容に修正を加えることなく、`#define` 文の部分を修正するだけで、収束条件を変更することが可能となる。この例では、収束判定を行なっているのが 1 箇所のみなので、大きな改善には見えないが、収束判定が多く箇所で行われるような大規模なプログラムに発展した際には、プログラミングミス（バグ）を防ぐ重要な役割を果たす。

図 4.3 のプログラムでは、 $C > 0$  が入力された場合には問題ないが、このアルゴリズムは  $f(x)$  の  $x$  切片が存在する場合のものであるから、もし  $C \leq 0$  の値が入力された場合にはその動作が保証できない。実際  $C < 0$  の値が入力されると、無限ループに落ちいり、正常な結果がえられない。これでは、正しく動作するプログラムとは言えない。

### goto 文を用いた処理の流れが分かりにくい例

そこで、 $C < 0$  の場合の問題を避けるために、以下のように `goto` 文を用いることもできる。（15, 24 行目）しかし、`goto` 文は `while` 文をまたいでおり、わかりづらい。この程度の長さのプログラムなら、24 行目が飛んだ先だと簡単に認識できるが、100 行～1000 行先、つまりひと目で認識できない先に処理が飛んだ場合、データの流れを認識するのに混乱をきたす原因となる。なおかつ、24 行目は  $C > 0$  の場合にも実行されるので、適切な解決とは言えない。

### 処理の流れを明瞭にする条件分岐の例

図 4.5 に、`if` 文だけを用いて  $C < 0$  の場合の問題を避けたプログラムを示した。このようなアルゴリズム構造にすることによって、処理の流れはを見失うことなくなおかつ、インデントを用いることで、その構造を把握することも容易となる。すなわち、バグが発生する可能性を小さくすることが出来る。

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main() {
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    if(C<=0) goto end;
16    x=C;
17    dx=-(x*x-C)/(2.0*x);
18    while(fabs(dx)>eps) {
19        x=x+dx;
20        printf("%12.10f \n",x);
21        dx=-(x*x-C)/(2.0*x);
22    }
23
24    end: printf("%12.10f \n",x);
25
26 }
```

図 4.4: `goto` 文を用いて  $C < 0$  の場合の問題を避けたプログラム。処理の流れを分かりづらくする可能性がある。

### 関数をつかう例

図 4.6 に、関数を用いたプログラムを示す。関数を用いることのメリットは、同じ処理を何度もプログラム中に記述することを避けるためであったり、あるいは、複雑な一連の処理をひとかたまりとして抽象化できることである。この例のようにアルゴリズム自体がもともと単純な場合には、それらのメリットはあまりないかもしれない。それでもこのように関数を用いると、数学的な表式とプログラム中の関数の名前を類似したものにすることによって、バグが生じる危険性を軽減できる。また、 $f(x) = x^2 - C$  以外の関数にたいして、ニュートン・ラフソン法を適用する場合にも、`main` 関数の部分を直接変更することなく、関数の内部だけを修正することによって、変更を済ませることができる。

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 int main(){
6
7     double C;
8     double dx;
9     double x;
10
11    printf("C=?");
12    scanf("%lf",&C);
13    printf("%lf\n",C,eps);
14
15    if(C<=0){
16        printf("C is zero or negative.\n");
17    }else{
18        x=C;
19        dx=- (x*x-C) / (2.0*x);
20        while(fabs(dx)>eps){
21            x=x+dx;
22            printf("%12.10f \n",x);
23            dx=- (x*x-C) / (2.0*x);
24        }
25    }
26}
27

```

図 4.5: if 文だけを用いて  $C < 0$  の場合の問題を避けたプログラム。while 文をまたいで、処理を遠くに飛ばさない。また、インデントを用いて、処理の構造を視覚的にとらえやすくする。

## 4.4 再利用して発展できるコード

図 4.2 に示されたコード（プログラム）と、図 4.6 に示されたコードを見比べると前者の方が行数が少なくまとまっているので、一見、簡潔で分かりやすいプログラムに見えるかもしれない。どちらがよりデータの流れ、および処理の構造を理解しやすいかよく見比べて欲しい。

コードを書いた時点では、データの流れや処理の構造は脳の中にイメージされている。両者の違いは理解できないかもしれない。むしろ前者のコードの方が理解しやすいと感じるかもしれない。

しかし、プログラムソースコードは使い捨てされると限らない。むしろ数ヶ月あるいは数年後に、そのコードを再び利用することがある

---

```

1 #include <stdio.h>
2 #include <math.h>
3 #define eps 1.0e-6
4
5 double Func(double x, double C);
6 double dFunc(double x);
7
8 int main(){
9
10    double C;
11    double dx;
12    double x;
13
14    printf("C=?");
15    scanf("%lf",&C);
16    printf("%lf\n",C,eps);
17
18    if(C<=0){
19        printf("C is zero or negative.\n");
20    }else{
21        x=C;
22        dx=-Func(x,C)/dFunc(x);
23        while(fabs(dx)>eps){
24            x=x+dx;
25            printf("%12.10f \n",x);
26            dx=-Func(x,C)/dFunc(x);
27        }
28    }
29
30 }
31
32 double Func(double x, double C){
33     double y;
34     y= x*x - C;
35     return y;
36 }
37
38 double dFunc(double x){
39     double y;
40     y=2.0*x;
41     return y;
42 }
```

図 4.6: 関数を用いてプログラムを記述すると、数学的表式と類似性があるので、バグの防止に役立つ。また、異なる関数についてのプログラムを作成するばあいに、メインルーチンを修正せずに済む。

であろう。あるいは改良を加えて他のプログラムの一部として活用するかもしれない。時間を置いた再利用ではないとしても、コードを書いた本人以外がそれを利用する場合もある。

そのような状況においても、データの流れと処理の構造が理解しやすいということは、短い行数で書かれているということよりも、本質的に重要である。

また、新しく作られるコードが、正しく動作するプログラムであるためにも、このことは必要である。正しく動作しないコードを一部にでも含むプログラムは、全体として正しく動作するプログラムとは言えないものである。

## 4.5 円周率 $\pi$ の近似値を求める

半径 1 の円の円周の長さは  $2\pi$  である。すなわち、この長さを  $L_c$  とすると、 $L_c = 2\pi$  である。

次に、この円に内接する正  $2^{n+1}$  角形 ( $n = 1, 2, \dots$ ) を考える(図 4.7 参照)。この多角形の一辺の長さを  $L_n$  とする。

この多角形は  $n$  を大きくしていくと、円に近づく。したがって、辺の長さ  $L = 2^{n+1}L_n$  は  $L_c$  に近づく。半径 1 の円周の長さを 2 で割ったものが  $\pi$  であるから、 $2^n L_n$  の値が  $\pi$  の近似値を与える。

では、具体的に  $L_n$  の値はどのように求められるだろうか？ $n+1$  角形の一辺の長さは、 $n$  角形の一辺の長さから求めることができる。つまり、 $n=1$  の 4 角形から順番に 8 角形、16 角形、32 角形…と求めていくことができる。

図 4.7 とピタゴラスの定理から、

$$x^2 + \left(\frac{L_1}{2}\right)^2 = 1 \quad (4.5)$$

$$y^2 + \left(\frac{L_1}{2}\right)^2 = L_2^2 \quad (4.6)$$

$$x + y = 1 \quad (4.7)$$

図 4.7: 半径 1 の円と、それに内接する正  $2^{n+1}$  角形 ( $n = 1, 2, \dots$ )

である。これは  $n = 1$  に対する関係であるが、一般の  $n$  に対しても

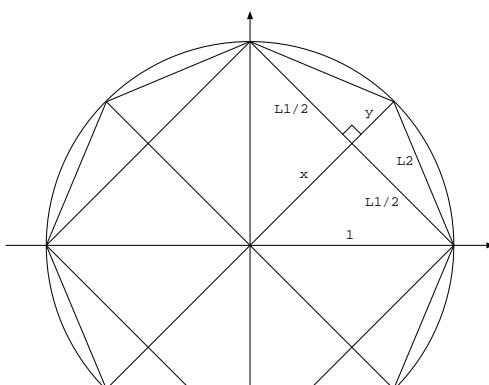
$$x^2 + \left(\frac{L_n}{2}\right)^2 = 1 \quad (4.8)$$

$$y^2 + \left(\frac{L_n}{2}\right)^2 = L_{n+1}^2 \quad (4.9)$$

$$x + y = 1 \quad (4.10)$$

が成り立つ。

これら 3 つの式から、 $x, y$  を消して、 $L_{n+1}$  と  $L_n$  の関係式を導けば、円に内接する多角形の一辺の長さ  $L_n$  に関する漸化式が得られる。



## 課題

1.  $x, y$  を消去することにより,  $L_n$  の漸化式を求めなさい.
2.  $L_n$  の漸化式を用いて  $\pi$  の近似値を求めなさい.  
その際、前節でつくった平方根のプログラムを関数として用いること.
3. 求められた  $\pi$  の近似値を  $n$  の関数としてグラフにすること(付録??参照).  
漸化式の繰り返しを多くしすぎると、誤差により正確に  $\pi$  の近似値が求められないので、注意すること.