第1章　数学ソフトウェアの現状と数値計算の役割について

　そして石井は，尾見半左右の文章の中の数値解析という言葉を指して,「池田さんは数値解析の本をやたらに持っていて，数値解析には尋常ならざる熱の入れ方だった」と語った。
　だが，数値解析と突然いわれても私にはさっぱり理解出来ず，あらためて数値解析について問い直さねばならなかった。
「数値解析というのは，要するに，すべてを近似値に計算するのです。微分とか積分といった高級なものを使わないで，全部加減乗除になおしてやってしまう。あるいは足し算，引き算でやってしまうわけです。」

　田原総一郎「日本コンピュータの黎明」（文藝春秋）

1.1　数学と計算

　大部分の人々は，多分，

$$\text{数学} = \text{計算} \tag{1.1}$$

なんて思っているんじゃないだろうか。
　しかし，教養課程で古き良き時代の微分積分や線型代数（あるいはそれに類似した科目）を叩き込まれた人ならば，(1.1) 式は首肯し難いに違いない。「計算が数学を構成している一要素であることは間違いない事実だが，せめて

$$\text{数学} \supset \text{計算} \tag{1.2}$$

と書くべきだ。」そう反論するに違いない。
　どちらが正しいのか，ということを言い出すとこれは哲学論争になる。数学及びその関連分野を自分の専門と自負している人にとっては,「いろいろな考え方があってよい」といった甘っちょろいものではない。それは自分の存在理由の根幹をなすものだったりするので，迂闊に結論が出せるものではない。万人が納得する結論を得ること自体無理がある。
　では計算とは何だろう？
　これは割と簡単に答えが出る。

> けいさん【計算】
> 1. 数量を計ること。はかりかぞえること。勘定すること。＊三国伝記 - 一一・二「殺生は都て計算せず」
> 2. 見積ること。予想すること。「計算に入れる」

> 3. あたえられた数、量、式を一定の規則に従って処理すること。また、未知の数、量、式を公式などを用い、演算の規則に従って求めること。「円周率の計算」
>
> 「国語大辞典（新装版）」(小学館) より。

極めて抽象的な物言いであるが，小中高で学んできた全ての「計算」を説明しようとするとこのように言わざるを得ないだろう。特に数学で学んできた「計算」は 3 の意味で使われている。

しかしこの説明も完全ではない。特に 3 の説明の後半「演算の規則に従って」の「演算」は，同じ辞書を引けば

> えんざん【演算】
> 計算すること。運算。

と「計算」という言葉を使ってしまっている。これでは説明にならない。本書では,「演算 (arithmetic)」を,「最もプリミティブな計算」，例えば四則「演算」のように狭くとらえることにし,「計算」については次のように定義する。

定義 1.1.1 (計算 (Calculation))
与えられた数，量，式を一定の規則に従って処理すること。また、未知の数，量，式を公式，アルゴリズム (算法) に則り，論理的な繋がりを考慮しながら求めること。

前半の定義は例えば「$1+(-3)=-2$」「$(x+1)^3+4x+1=x^3+3x^2+7x+2$」というような直線的なものを，後半の定義は「$\square+(-3)=-2$ となる数を求めよ。」「$x^3+3x^2+7x+2=(x+1)^3+(4x+1)$」というような，前者のような計算の結果から演繹することを求められるものを指していると考えて良い。

一般に，前者より後者のような問題は難しいと思われている。その理由として，前者の計算を散々経験し，その経験の中から後者の問題を考える糸口を見つけ出す必要があるからだろう。小学校でもまず低学年で自然数の四則演算 (九九も含む) や分数・小数の扱いを学んだ上で，ごく簡単な一次方程式を解くような問題を高学年で扱うようになっている。中学校でも，まず文字式，特に多項式の演算を練習した上で，因数分解を習うように教科書は構成されている。高校でも事情は同じであろう。

大学でも全く同じで，高校で数学 I・II・III・A・B・C を全て学んできたという前提に立って教育を行いたい所であるが，実際はこの程度の内容では全く足りない。特に数学を世界共通の言語として頻繁に使用する分野では，微分積分の内容に限っても高校での内容ではお話にならない。級数を扱わない，Taylor 展開には触れない，逆三角関数も出てこなければ，無限積分もない。二以上の変数を持つ関数は全く扱わず，当然偏微分も重積分もない。そのために，数学を必要とする学科ではわざわざ一変数の微分積分からやり直している有様である。近年特に内容が削られたこともあって，ベクトルや行列については全く知らないというケースが多く，線型代数を担当する真面目な教師は結構な苦労を強いられることになる。

しかし，とりあえずは大学教養程度の「微分積分」「線型代数」を終えれば,「未知の数，量，式を公式などを用い，論理的な繋がり考慮しながら求める」計算も，その対象を広げることができるようになる。その上に立ってはじめてこれから学ぶ「数値計算」が理解できるようになるのである。

では「数値計算」とはいかなる計算を指す言葉なのか？ 以下，このことについて考えることにする。

1.2 数値計算とは？

数値計算とは，Numerical Computation の和訳である。文字通りだと「数の計算」ということになるが，今では一つの学問分野を表わす単語として使用されることが多い。特に「数値計算」についての研究一般については「数値解析 (Numerical Analysis)」という名前もよく使用する。本書ではどちらかといえば親しみの湧きそうな前者をタイトルに使っているが，個人的にはあまり考慮せずにその場の雰囲気に応じて適当に使い分けている。但し，計算そのものを表わすときには後者の名称は使いづらい。このような両者をごっちゃにすることは，あくまで学問分野の名称としてのみ許されることである。

数値計算・数値解析はどのような学問か。ここでは Trefethen[40] の定義を引用しておこう。

定義 1.2.1 (数値解析 (Numerical Analysis))
数値解析とは，連続数学 (continuous mathematics) の問題に対するアルゴリズムの研究である。

これでは何のことか分からないかもしれない。具体的な問題 [12] で考えることにする。

例題 1.2.2 (ある算額の問題)
東京都浅草の鳥越社に奉納された算額 (文政 4 年，1821 年) に次の問題がある。単位の寸を省略し，現代風に書き直す。

長軸が 5, 短軸が 3 の楕円において，長軸に平行な弦の長さを 4 とする。この弦が楕円から切り取る弧のうち，短い方の長さ l はいくらか？

これが所謂連続数学の問題の一例である。早速解いてみることにする。曲線の長さを求める問題であるから，楕円 (長軸の長さを $2a$，短軸の長さを $2b$ とする) の媒介変数表示

$$\begin{cases} x &= a\sin\theta \\ y &= b\cos\theta \end{cases}$$

を使って

$$\begin{aligned} \int_\alpha^\beta \sqrt{\left(\frac{dx}{d\theta}\right)^2 + \left(\frac{dy}{d\theta}\right)^2}\, d\theta &= \int_\alpha^\beta \sqrt{a^2\cos^2\theta + b^2\sin^2\theta}\, d\theta \\ &= a\int_\alpha^\beta \sqrt{1 - \left(1 - \frac{b^2}{a^2}\right)\sin^2\theta}\, d\theta \end{aligned}$$

を得る。この問題の場合 $a = 5/2, b = 3/2$，$\alpha = 0, \beta = \sin^{-1} 4/5$ とした時の弧の長さの 2 倍となるから

$$l = 2\cdot\frac{5}{2}\int_0^{\sin^{-1}4/5} \sqrt{1 - \frac{16}{25}\sin^2\theta}\, d\theta = \int_0^k \frac{5\sqrt{1-k^2x^2}}{\sqrt{1-x^2}} dx \tag{1.3}$$

を得る。ここで $x = \sin\theta, k = 4/5$ である。

最後の定積分 (1.3) を計算すれば完了する。

さて，定積分は通常次のように行われる。関数 $F(x)$, $f(x)$ が

$$\frac{dF(x)}{dx} = f(x)$$

という関係にあるとき，定積分 $\int_\alpha^\beta f(x)dx$ は

$$\int_\alpha^\beta f(x)dx = F(\beta) - F(\alpha)$$

と計算される。が，この積分の場合，$F(x)$ を初等関数の有限個の組合せで表わすことができない。しかし，定積分の値は存在しないのかと言えばそうではない。積分区間内で $f(x)$ は連続であることは明らかで，その際には定積分の値は一つに確定する。

ではどうするか？

定積分の計算を「正確に」行うことを諦めて,「おおよその値」を求めるように方針転換を行えばよい。定積分 $\int_\alpha^\beta f(x)dx$ が存在しているのであれば，閉区間 $[\alpha,\beta]$ を n 分割し，各分点を $\alpha = x_0, x_1, ..., x_n = \beta$ とすれば

$$\lim_{n\to\infty} \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i) = \int_\alpha^\beta f(x)dx$$

となる。無限に細かく短冊に切り，各短冊 (長方形) の面積を足しこんでいけば，それが定積分の値になる。従って,「無限に」細かくすることは不可能であるが,「ある程度沢山」細かくしていけば次第に定積分の値に近づいていくことが期待される。つまり $n >> 1$ であれば

$$\int_\alpha^\beta f(x)dx \approx \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i) \tag{1.4}$$

と考えてよい。ここで ≈ は左右両辺の値が近いということを表わしている。

区間を 5, 10, ..., 500 に等分割し，(1.4) 式の右辺の有限和を計算してみると，次のようになる。

分割数	有限和の値
5	4.242281574
10	4.248617984
20	4.250309 24
40	4.250740169
100	4.250861447
500	4.250883653

従って，この定積分の値は I = 4.2508...(正確には 4.250884578881844496425748351167 1...) であろうと推察される。

このように，

1. 定積分のような連続的な問題に対しては「離散的 (discrete)」な「近似 (approximation)」を考え
2. 有限回の演算を行うことで解決を図る方法 (アルゴリズム) を探り

3. 実際に計算し，計算結果を検証する

ことを研究する学問を「数値解析」と呼び，この際に行われる計算を「数値計算」と呼ぶ。近似を伴う場合は，それがどの程度真の値と食い違うのか，つまり誤差 (error) はどの程度か，という考察も必要である。

但し，このような計算は人力では限りがある。実際，上の表は表計算ソフトウェア (Excel 2000) を用いて計算したものである。現在はパソコン (以下 PC と略記する) が満ち溢れており，このような単純計算は PC を利用するのが普通である。従って現在では「数値計算」は，PC 等の電子計算機 (以下，コンピュータ) 上で行われる上記のような計算という意味で用いられている。

一部の書籍では，ENIAC のような Neumann 型コンピュータの始まりを数値計算の始まりとしているが，それは少し偏狭的な見方といわざるを得ない。確かに J. von Neumann はコンピュータ上で実行される有限桁浮動小数点演算がもたらす誤差の影響 (第 3 章参照) を調べた最初の研究者であるが，上記の江戸時代の例を見ても分かるとおり，もともと数学，特に実用レベルの計算においては近似を伴う「数値計算」が人力によって広く行われていたのである。より正確を期すならば，そのような「古典的」数値計算の伝統と蓄積が，コンピュータのような自動計算機にジャストフィットするように改良を加えられて，今のような「現代的」数値計算になっていった，と言うべきである。

問題 1.2.1
定積分 (1.3) の近似値を (1.4) 式の右辺の有限和を用いて表計算ソフトウェアで計算する方法を解説し，実際に計算してみよ。

1.3 コンピュータ言語と数値計算

数値計算は Fortran で記述されたプログラムで行われるものだ，という時代が長く続いたせいか，近年のように様々な言語が登場するようになると，数値計算用に書かれた Fortran プログラムを単純に他言語に書き直しただけの安易な解説書が大量に出回るようになった。こういう風潮には正直あまり感心しないのだが，執筆者にとってはお手軽な題材であると同時に，自分の知識の範囲内で他言語の習得が出来，読者にとってもなるべく新しい言語で勉強した方が得であるという意識があるのだろう。出版社にとっても，売れると考えるから出版するのであって，いくら玄人筋に評価の高い本を出しても軒並み採算割れ，というのでは会社の存続も危うい。「安易な本」が出回るにも，それ相応の理由があってのことぐらいは承知しているつもりである。

そういう「○○ (←適当な言語名やソフトウェア名を当てはめよ) による数値計算」的な書籍の氾濫を前にすると，何を使って数値計算を勉強したらよいのか分からなくなる人もいるだろう。前節の数値例も「Excel による数値計算」[1]の一例である。個人的には，どれでも自分の現在のスキルと相談の上，相性が良さそうな本を兎も角一冊選んで一通り眺めてみればよい，と考える。他の分野は知らないが，少なくとも数値計算を名乗る本にトンデモなものがあるとは思われない。

統合型の数学ソフトウェアについては後述するが，シミュレーションにかけたいモデルが既に存在していてその結果にのみ興味がある場合，過去の資産もスキルもないのに一からプログラミング言語でシコシコ作り上げるというのもかなりの手間である。入出力のインターフェースの使いやす

[1]そんな名前の本もあった。

さを考えれば，まずは MATLAB/Scilab/Octave や Mathematica/MuPAD/Maple 等の数学ソフトウェアに触れてみることをお勧めしたい。

しかし，私はそれなりに「プログラミング言語でシコシコ」するスキルもある程度は磨いておいても損はしないと断言する。この場合の言語は Perl 等のインタプリタ主体のスクリプト言語ではなく，コンパイラで直接 CPU が処理できる機械語コードを掃き出すことのできる C/C++や Fortran, Pascal 等の言語を指す。

前節で取り上げた楕円積分を計算する FORTRAN77 プログラム例 (図 1.1) と，C プログラム例 (図 1.2) をここに掲載する。本書はプログラミングの本ではないので，文法については他の書籍を参照されたい。が，処理内容はどちらも同じで，

(3) 被積分関数の値を計算する部分 図 1.1 の 43〜50 行目, 図 1.2 の 4〜7 行目

(2) 積分の計算 (和の計算) を行っている部分 図 1.1 の 18〜39 行目, 図 1.2 の 9〜28 行目

(1) 主処理部 図 1.1 の 1〜14 行目, 図 1.2 の 30〜40 行目

となる。(1) がまず実行され，そこから (2) が呼び出されて積分計算が実行される。その際に，(2) へ引数として渡された (3) が必要に応じて呼び出されている，というイメージを思い描いて貰えればよい。

私は数値計算を学ぶ上で，プログラミング言語を使いこなすスキルを持つことのメリットとして，次の 2 点を挙げたい。

ブラックボックスの内部構造を類推できるようになる

言語の文法を理解していないと，処理の詳細を把握することは難しい。しかし，このようなソースコードが公開されていれば，そして自分である程度記述することが出来れば，初めからソースコードが公開されていない統合型の数学ソフトウェアだけを使用していたのでは分からない情報を得ることができる。統合型の数学ソフトウェアも，元はプログラミング言語で記述されたソースコードの塊である。ブラックボックス化されたものを扱う上でも，内部構造を類推する上でプログラミング言語を扱えるスキルがあることが望ましいのである。

プログラムのチューニングが出来るようになる

統合型の数学ソフトウェアの場合，以前よりかなり改善されたとはいえ，実行速度の面でまだ問題がある。自分の実行したいアルゴリズムに不必要な部分を呼び出すオーバーヘッドを極力減らし，さらに高速な数値計算ライブラリを利用したりする自由度を確保するにはプログラミング言語を習得するより他に手段がない。

他にも，あまり合理的な理由とは言いがたいが，精神的な自由度が違うことを個人的に挙げたい。商用であれフリーであれ，その統合型数学ソフトウェアでしか動かないスクリプトを作ってしまうと，どうしても実行環境に制限が掛かってしまう。将来的にそのソフトウェアを使うことが出来るかどうかも定かではない。その点，国際規格として完成されているプログラミング言語であれば，ソースコードは大抵の開発環境で再利用可能である。この気楽さは何者にも代え難い。

プログラミング言語は時と共に変化していくものであるが，一度培った上記のようなプログラミング上のスキルは文法上の相違を超えて生かすことが出来るのである。

```
 1:        INTEGER NUM_DIV
 2:        DOUBLE PRECISION A, B, F, INTEGRAL
 3:        EXTERNAL F, INTEGRAL
 4:C
 5:        A = 0.0
 6:        B = 0.8
 7:        NUM_DIV = 10
 8:C
 9:        WRITE(6,600) A, B, INTEGRAL(A, B, F, NUM_DIV)
10:C
11:600     FORMAT(1H ,'Integral[',F10.3,',',F10.3,']: ',E25.17)
12:C
13:        STOP
14:        END
15:C
16:C
17:C
18:        REAL*8 FUNCTION INTEGRAL(X_START, X_END, FUNC, NUM_DIV)
19:C
20:        DOUBLE PRECISION X_START, X_END, FUNC
21:        INTEGER NUM_DIV
22:        EXTERNAL FUNC
23:C
24:        DOUBLE PRECISION H, X, X_NEXT
25:        INTEGER I
26:C
27:        INTEGRAL = 0.0
28:        H = (X_END - X_START) / NUM_DIV
29:        X = X_START
30:        X_NEXT = X + H
31:C
32:        DO 1000 I = 1, NUM_DIV
33:                INTEGRAL = INTEGRAL + FUNC((X + X_NEXT) / 2) * H
34:                X = X_NEXT
35:                X_NEXT = X + H
36:1000    CONTINUE
37:C
38:        RETURN
39:        END
40:C
41:C
42:C
43:        REAL*8 FUNCTION F(X)
44:C
45:        DOUBLE PRECISION X
46:C
47:        F = 5.0 * SQRT(1.0 - (0.8*x)**2) / SQRT(1.0 - x**2)
48:        RETURN
49:C
50:        END
```

図 1.1: FORTRAN77 のプログラム例

```
 1:#include <stdio.h>
 2:#include <math.h>
 3:
 4:double f(double x)
 5:{
 6:        return 5.0 * sqrt(1.0 - 0.8*0.8*x*x) / sqrt(1.0 - x * x);
 7:}
 8:
 9:double integral(double x_start, double x_end, double (*func)(double x), \
10:long int num_div)
11:{
12:        double ret, x, x_next, h;
13:        long int i;
14:
15:        ret = 0;
16:        h = (x_end - x_start) / num_div;
17:        x = x_start;
18:        x_next = x + h;
19:
20:        for(i = 0; i < num_div; i++)
21:        {
22:                ret += func((x + x_next) / 2) * h;
23:                x = x_next;
24:                x_next = x + h;
25:        }
26:
27:        return ret;
28:}
29:
30:main()
31:{
32:        long int num_div;
33:        double a, b;
34:
35:        a = 0.0;
36:        b = 0.8;
37:        num_div = 10;
38:
39:        printf("Integral[%f, %f] : %25.17e\n", a, b, integral(a, b, f, num_div));
40:}
```

図 1.2: C のプログラム例

1.4 数学ソフトウェアについて

「数学ソフトウェア」という言葉はかなり漠然としたものであるが，ここでは “Mathematica™” や “MATLAB™”のような対話型インターフェースを持った統合型のソフトウェアについて考える。

まず，先の節で定義した言葉として「数値解析」がある。もう少し説明を付加すると,「狭義の」数値解析とは，数としては CPU やその周辺の回路で直接サポートしている整数や有限桁の浮動小数点数 (後述) を用い,「すべてを近似値」で行われる計算を研究対象とする学問である。計算そのものは「数値計算」という名称で呼ばれる，ということは既に述べた。

それに対して，人間が普段行う手計算のように

$$p(x) = 3x^2 - 5x + 1$$

という多項式に対して

$$\begin{aligned} p'(x) &= 6x - 5 \\ \int p(x)dx &= x^3 - \frac{5}{2}x^2 + x + C \end{aligned}$$

と微分は微分のまま，積分は積分のまま計算した「式」を求めたい場合に用いられるソフトウェアを総称して「数式処理」ソフトウェアと呼び，そのような処理そのものは「記号処理」と呼ばれることが多い。数式処理ソフトウェアとしては，古くは MACSYMA, Reduce 等がある。満足なハードウェアの性能が得られなかった時代は，正直言ってあまり使いやすいものとは思われなかった。扱うものを多項式に限っても，計算が複雑になり項数が多くなると OS ごとマシンがフリーズしてしまうし，出力される数式も読みやすいものではなかった。そのため，数式処理ソフトウェアは研究者のためのもの，というイメージが長く定着してしまったように思われる。

そのような状況が一変したのは，Mathematica が登場してからである。Mathematica は使いやすい GUI を搭載し，グラフィックの機能も備え，音声まで扱うことが出来る。何より画期的だったのは，ハードウェアでサポートする範囲の浮動小数点数を使った数値計算の機能も搭載していたことである。そのため，ソフトウェアでのみ浮動小数点数を扱っていた[2]従来の数式処理ソフトウェアよりも格段に速い計算が可能になった。以降，その他の数式処理ソフトウェアは Mathematica 同様，GUI を備え，グラフィックスの機能を持ち，浮動小数点数による高速な数値計算の機能もある程度はサポートするようになった。

MATLAB はこの点，出発点が違っていた。MATLAB の名前は，MATrix LABoratory の略称であることから分かるように，元々は浮動小数点数による線型計算を簡単に実行できるように対話形式のインターフェースを装備したのが発端であり，ベースは全て数値計算のライブラリ，特に LAPACK である。故に，MATLAB は「数値計算ソフトウェア」と言える。現在ではグラフ作成機能や文書処理ができるようになってはいるが，計算機能の根本はあまり変化していないようである。MATLAB の計算機能をかなりの部分サポートしているフリーソフトウェアとして Scilab や Octave があるが，これはソースコードも公開されており，自由に読むことが出来る。これを眺めてみれば，

[2]ハードウェアで直接扱うことの出来る浮動小数点数よりも仮数部桁数が多いもの (これを多倍長浮動小数点数と呼ぶ (後述)) を利用できる，というメリットはあったが，計算時間が長くなるという難点がある。

インターフェース周りを除いて，現在まで培われてきた信頼ある数値計算ライブラリの集積で成り立っていることが理解できる。

このように，GUI や様々な機能拡張によって一見判別しづらくなってはいるが，Mathematica に代表される「数式処理ソフトウェア」と MATLAB に代表される「数値計算ソフトウェア」の基盤はそれぞれ異なっている。以上をまとめたのが図 1.3 である。

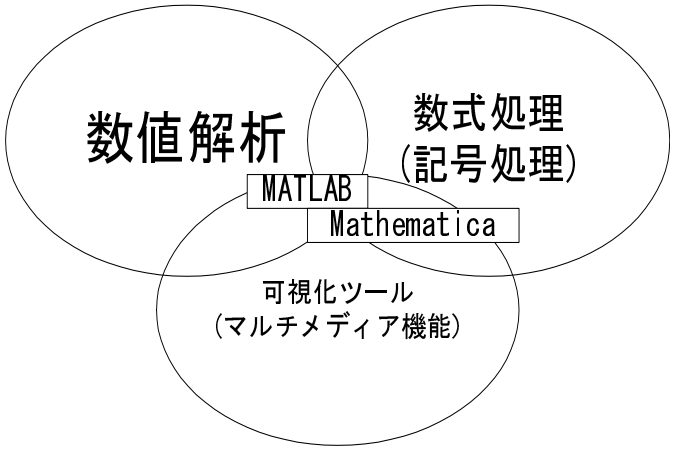

図 1.3: 数学ソフトウェアの概念図

もっとも，Mathematica や MATLAB が利用者を増やしたのは，数値計算ライブラリの整備が進み，数式処理の研究が進んだためだけではない。OS 自身が GUI を搭載し，開発環境が提供され，更にそれだけの規模のソフトウェアをスムーズに動作させるパワーを持ったハードウェアも必要である。俗っぽい表現を使えば「時代がそのようなソフトウェアを出現させた」と言える。

最後に，図 1.4 で例題 1.2.2 の積分を，Mathematica と Octave で実行する方法及びその結果を載せておく。使い方の詳細については付属のマニュアル等を適宜参照されたい。

1.5　数値計算実行方法の相違

プログラム言語から呼び出して使用するソフトウェアライブラリの場合，コンパイラを使ってソースコードから単独で実行可能な実行プログラムを生成し，数値計算の結果はこの実行プログラムが吐き出す出力として得る。ライブラリに関数もしくはサブルーチンとして登録された機能は，コンパイル時にリンクされて実行プログラム内に組み込まれる (静的リンクの場合[3])。もし，実行結果に問題が発生した時には，デバッグ作業を行い，ソースコードを修正して再度同じ手順を実行する (Fig.1.5)。

これに対して，Mathematica や MATLAB といった統合型の数学ソフトウェアの場合，ソースコードは「スクリプト」という，直接，逐次的に数学ソフトウェアによって解釈され実行可能なプログ

[3]実行プログラムが走って必要となった時に呼び出される，という動的リンク機能を用いることもある。この場合，コンパイル時には動的にライブラリを呼び出すインターフェース部分のみがリンクされる。同じライブラリを多数の実行プログラムが参照する時，プログラムサイズが小さくて済むという利点がある。

```
In[13]:= Integrate[5 * (1-(4/5)^2*x
^2)^(1/2)/(1-x^2)^(1/2), {x, 0, 4/5
}] ← 記号処理で積分を行おうとすると
 ...

Out[13]= \!\(5\ EllipticE[ArcSin[4\
/5], 16\/25]\) ← これ以上の計算不能

In[12]:= NIntegrate[5 * (1-(4/5)^2*
x^2)^(1/2)/(1-x^2)^(1/2), {x, 0, 4/
5}] ← 数値計算で積分を実行すると・
 ..

Out[12]= 4.25088 ← 定積分の値が出る
```

```
# Sample Script for Octave

function y = f(x)
        y = 5 .* sqrt(1 - 0.8^2 .*
x^2) / sqrt(1 - x^2);
endfunction

a = 0.0;
b = 0.8;

[val, ierror, nfunc, error] = quad(
"f", a, b);
printf("Integral[%f, %f] : %25.17e\
n", a, b, val);

実行結果:
Integral[0.000000, 0.800000] :   4.
25088457888184479e+00
```

図 1.4: Mathematica(左) と Octave(右) の例

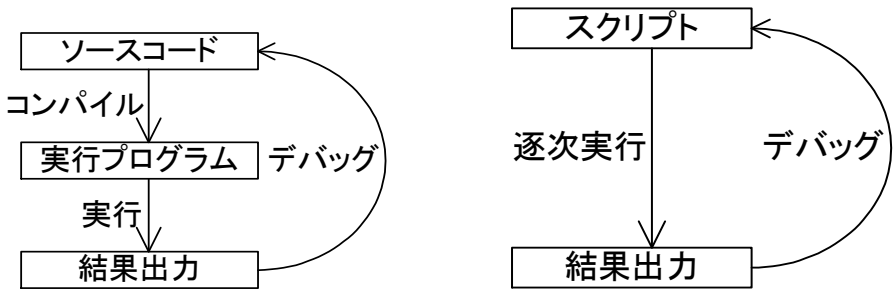

図 1.5: 実行手順:(左) ライブラリ利用, (右) 統合型ソフトウェア利用

ラムの形式でユーザが作成し，計算結果は即座に得られる。デバッグ作業が必要な時はスクリプトを修正し，再度逐次実行させればよい[4]。ソフトウェアライブラリを用いる場合に比べ，デバッグしてから結果を再度得るまでの手順が簡潔で，試行錯誤を重ねる段階では利便性が高い。反面，構文解析を行いつつ逐次実行されるため，計算速度が実行プログラムのそれより若干低下することが多い。

参考図書

現在は「数値計算」という狭義の言葉を使うよりも,「科学技術計算 (Scientific Computing)」というより広義の言葉を使うことが増えてきたようである。数値計算という資産を活用して応用的な処理を行うことに重点が移ってきたという時代背景もある。従って，現在のコンピュータ環境，特に数値計算を必要とする分野のものを知るためにはこの言葉をサーチすればよいということになる。

最高性能のコンピュータ環境の動向を知るには，

TOP 500 Supercomputer Sites
http://www.top500.org/

が一番参考になる。その中でも，PC を多数ネットワークに繋げて分散的に並列処理を行う PC Cluster が登場してきていることは注目に値する。数台のレベルであれば，数十万円程度の予算でハードウェアを揃えることが出来，Linux 等の Free OS とプログラム開発環境をを利用すればその費用は不要である。但し，実際に使いこなすには相当のプログラミングスキルがないと難しい。単一 CPU のプログラムとは全く違うアプローチを必要とするからである。

演習問題

1. 次の定積分の値を求めよ。どんな方法・ソフトウェアを用いてもよい。

$$\int_0^{\pi/2} \cos(\sin x)\, dx$$

2. 表計算 (Excel, OpenOffice calc 等), 数式処理 (Mathematica, MuPAD, Maple 等)，数値計算 (MATLAB, Scilab, Octave 等), コンパイラ (Fortran, C, C++, Java 等), スクリプト言語 (Perl, Ruby 等) の，浮動小数点演算やそれ以外の機能を調べ，その優劣を論じよ。

3. 数値計算において特に重視される要素は

   (a) 求めたい近似値と真の値との「近さ」··· 精度

   (b) 近似値を求めるのに要する「時間」··· 計算時間

   の二点である。本書で扱っている各種のアルゴリズムを，異なるコンピュータ環境 (ハードウェア，OS, 言語/アプリケーションソフト) において実行し，これらの二点を調査し，比較検討せよ。

[4] スクリプト言語と呼ばれるインタプリタを用いる場合も同様の実行手順となるが，数値計算の機能を補うためにはソフトウェアライブラリを併用する必要がある。