講座

プログラミングで学ぶ有限要素法

2. プログラミングの基礎

渦岡良介（うずおか　りょうすけ）
東北大学大学院准教授　工学研究科

宮田喜壽（みやた　よしひさ）
防衛大学校准教授　システム工学群

山川優樹（やまかわ　ゆうき）
東北大学大学院准教授　工学研究科

2.1 プログラミングをはじめる前に

初心者にとってプログラミングの最初の壁はカタカナ用語であろう。以下では図—2.1に示す流れに沿って，プログラミングの一般的な作業の流れの中でよく使う用語について説明する。用語のより詳しい説明は専門書やインターネットを参照されたい。

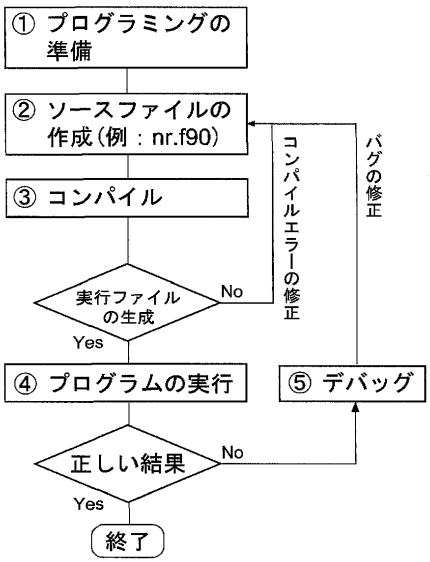
① プログラミングの準備

プログラミングとは計算式や計算の手順（アルゴリズムと呼ばれる）を記述したプログラムを作成することである。コンピュータが実行可能なファイル[注1)]をコンピュータの専門家でない人が直接プログラミングして，作成することは困難である。このため，計算式や計算の手順を普通の人が分かりやすく記述できように開発されたのがプログラミング言語である。プログラミングの第一歩がこのプログラミング言語を選択することである。

プログラミング言語には多くの種類があるが，科学技術計算でよく用いられるものはFORTRAN77，Fortran 90/95, C, Javaなどである。大型計算機が主流であった1980年代ではFORTRAN77が広く使われていたため，現在でも地盤工学関係のプログラムではFORTRAN77で記述されたものが多いが，本講座ではFortran90/95（以下，Fortran）を採用する。FortranではC, Javaと類似した文法や配列（ベクトルやマトリックスの値を格納する変数）演算が整備されており，FORTRAN77よりも格段に使いやすくなっている。なお，後のプログラム例では他のプログラミング言語でも基本は変わらないことを示すため，CおよびJavaのプログラム例も付録で示す。

コラム：プログラミング言語

最近はMATLABに代表されるような行列演算などの様々な数値解析プログラムがライブラリ（ユーザーが自由に組み込み可能なプログラム）として準備されているプログラミング言語も普及している。このライブラリを用いる場合，例えば連立一次方程式の求解などの数値計算に関するプログラミングは不要となるため，計算式や計算の手順のプログラミング作業に集中できる利点がある。いずれの言語を用いるにしても，基本的な計算の手順は変わらないので，言語の選択は好みの問題である。



図—2.1　プログラミングの基本的な流れ

② ソースファイルの作成

計算式や計算の手順をプログラミング言語の文法に基づいて記述したものはソースコードと呼ばれ，それをテキスト形式として保存したファイルはソースファイルと呼ばれる。ソースファイルはテキスト形式であるため，テキストエディタによって編集する。Fortranの場合，ファイルの拡張子をf90やf95とするのが一般的である。

③ コンパイル

コンパイルとはソースファイルをコンピュータが実行可能なファイルに変換することであり，この変換を行うソフトはコンパイラと呼ばれる。よって，プログラム言語によってコンパイラは異なり，Fortran言語ならFortranコンパイラ，C言語ならCコンパイラなどと呼ばれる。

プログラミング言語の文法上の間違いのようにコンパイラが指摘してくれる比較的分かりやすいプログラムの間違いは，コンパイル作業を繰返しながら修正する。

④ プログラムの実行

作成したプログラムの実行形式のファイルを実行する。

注1) 実行形式のファイル，Windows環境では拡張子がexeのファイル。

この際，プログラムは計算に必要となる入力データを記述したテキストファイルを読み込み，計算結果をファイルに出力する。

⑤　デバッグ

よほど注意力のある人でなければ，最初に書いたプログラムには必ず間違いがある。プログラムの間違いのことをバグと呼び，このバグを取る作業のことをデバッグと呼ぶ。思ったとおりの計算結果が出ない場合など計算式や計算の手順の入力ミスなどのバグは，プログラムの実行を繰返しながら修正する。この作業では，意図したとおりの計算が実行されているか確認するために，最終的に必要な計算結果だけでなく，計算途中の変数の値などをファイルに出力する作業を行う場合もある。

2.2　PCでプログラミングを行うために

前述のプログラミング作業の中で②，③および⑤においては特別のソフトが必要となることから，PC上に自分のプログラミング環境を構築する必要がある。ここでは，本講座で必要とするプログラミング環境について述べる。

2.2.1　プログラミング環境の種類

プログラミング環境には主に以下の二つの種類がある。

(1)　テキストエディタとコンパイラを利用

ソースファイルをテキストエディタで編集した後，コマンドを入力[注2]することでコンパイラを実行する。テキストエディタおよびコンパイラとも，商用ソフトから安価なシェアウェア，フリーソフトまで様々なソフトがある。シェアウェアやフリーソフトは商用ソフトと比較して使い勝手は劣る面があるが，PCさえあれば安価にプログラミング環境を構築することができる。本講座ではシェアウェアやフリーソフトを用いる方法を用いて，プログラミング環境を構築する。

テキストエディタについては，Windows環境における「メモ帳」でもよいが，プログラミング言語の予約語（文法上の決まり文句）などを強調表示できる機能などプログラミングに便利な機能が付いているテキストエディタ[注3]の方が使いやすい。読者は自分の好みのテキストエディタを利用して欲しい。コンパイラについては後述する。

(2)　統合開発環境（IDE）を利用

GUIを用いた統合開発環境（IDE）と呼ばれるソフトには，ソースファイルを編集するエディタの機能とコンパイルやデバッグなどの機能が組み込まれている。IDEではコマンドではなくアイコンやメニューからコンパイルが可能であり，コンパイル時の文法エラーや実行時のエラーなどを分かりやすく提示してくれる。商用のコンパイラは一般にIDEを装備しており使いやすくなっている。フリーソフトでもEclipseなどのIDEが利用できる。

2.2.2　コンパイラ

以下では，Fortran, C, Javaについて，Windows環境におけるコンパイラを紹介する。なお，フリーソフトだけでもいくつかのソフトがあるが，ここではWindows環境で比較的扱いやすいソフトを紹介する。なお，これらは2010年2月時点のものであり，URLなどは今後変更の可能性がある。これらの多くはUNIXから移植されたものであるので，Linux環境やMac環境の読者はより容易にコンパイラの導入が可能である。興味のある読者は自分の好きなコンパイラを利用して欲しい。

(1)　Fortran

ソフト名：g95
配布元：http://www.g95.org/
ダウンロード先：http://ftp.g95.org/g95-MinGW.exe
インストール方法：ダウンロードしたファイルを実行する。その後，コマンドプロンプトで，
＞g95
と入力して，“g95: no input file”と表示されればOK。表示されない場合はシステムの環境変数であるpathに“インストールフォルダ¥bin”[注4]が入力されていることを確認し，なければ追加する。また，同じく環境変数のLIBRARY_PATHに“インストールフォルダ¥lib”が入力されていることを確認し，なければ追加する。動作しない場合はg95のreadme.txtやマニュアルを参照して欲しい。なお，g95はFORTRAN77, Fortran90/95に対応している。

(2)　C

ソフト名：gcc
配布元：http://gcc.gnu.org/
ダウンロード先：http://sourceforge.net/projects/mingw/
インストール方法：ダウンロードしたファイルを実行した後，コマンドプロンプトで，
＞gcc
と入力して，“gcc: no input file”と表示されればOK。表示されない場合はシステムの環境変数のpathに“C:¥MinGW¥bin”が入力されていることを確認し，なければ追加する。gccはGNU Compiler Collection（GCC）と呼ばれるフリーのコンパイラソフト集の中核をなすCのコンパイラである。なお，GCCにはFORTRAN77, Fortran 90/95, Javaも含まれている。

(3)　Java

ソフト名：Java Development Kit（JDK）
配布元：http://java.sun.com/javase/
ダウンロード先：http://java.sun.com/javase/ja/6/download.html中のJDK
インストール方法：ダウンロードしたファイルを実行した後，コマンドプロンプトで，
＞javac
と入力して，“使い方：Javac 〈options〉〈source files〉...”

注2)　Windows環境ではコマンドプロンプトを用いる。画面上のボタンをクリックするGUIになれた読者には不慣れかもしれないが，使用するコマンドは限られているので是非トライして欲しい。

注3)　例えば，「秀丸」（http://hide.maruo.co.jp/）など。

注4)　例えば，“C:¥Program Files¥g95”にインストールした場合は“C:¥Program Files¥g95¥bin”。

と表示されればOK。表示されない場合はシステムの環境変数のpathに“C:¥Program Files¥Java¥jdk1.6.0_15¥bin”が入力されていることを確認し，なければ追加する。Javaを用いてプログラムを作成する環境はJDK，Javaで作成されたプログラムを実行する環境はJREと呼ばれる。

2.3　プログラムの基本

プログラムは一般に以下の①～⑥の要素で構成されている。ここでは，これらの要素の基本的な内容を説明し，その具体例は次節以降に示す。

①　プログラムの開始

プログラムの開始を表す記述である。プログラムの名前などの情報を付加する。

②　変数の定義

例えば，フックの法則$\sigma=E\varepsilon$を用いて，応力σを計算する場合，手書きのノートでは，

$E=1\,000$(kPa)

$\varepsilon=0.001$

$\sigma=1\,000\times0.001=1.0$(kPa)

と書くであろう。このノート中で，εおよびσが変数として用いられており，プログラム中の変数もこのノートの変数と同様なものである。ただし，プログラムでは$E=1\,000$として数値を代入する前に，変数Eがどのような種類の変数であるかをあらかじめ定義しておく必要がある。つまり，変数に数値を代入して変数に数値を記憶させる前に，記憶するための領域を準備しておくのである。この準備を行うのが変数の定義である。

③　計算条件の入力

計算に用いる数字をデータファイルから入力し，②で定義した変数に代入する。

④　計算式や計算の手順

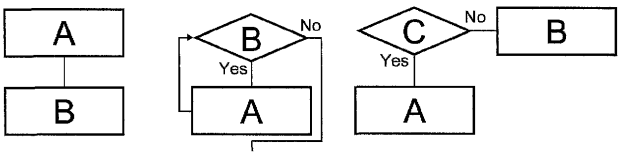
プログラムの中核部分であり，計算式や計算の手順を記述する。プログラムを記述する上での基本的な構造は**図—2.2**に示す1)順次，2)反復，3)分岐の三要素である。どのプログラミング言語を用いても基本はこの三要素である。

1)　順次とは，プログラムに記述された順に計算Aの次に計算Bを実行することである。図中の四角形は計算などの処理を，線は上の処理から下の処理への流れを示す。

2)　反復とは，一定の条件Bを満たす間，同じ計算Aを繰返し実行することである。図中のひし形は条件判断を示す。下から上など通常の流れと異なる流れの場合には線に矢印を付ける。

3)　分岐とは，ある条件Cを満足する場合は計算Aを実行し，満足しない場合は計算Bを実行することであり，条件によって計算方法を変えることである。

以上の順次，反復，分岐の三要素を用いて，計算の手順を具体的に記述したものがアルゴリズムであり，フローチャートによって示される。計算の手順に基づきこの三要素を用いて，正しい[注5]アルゴリズムを作成することがプログラミングの本質である。アルゴリズムを作成することによって，数式で記述されている抽象的な理論の計算方法を具体的に理解することができる点が，プログラミングを学習する意義の一つであり，このことが本講座の目的の一つである。



1)　順次　　2)　反復　　3)　分岐

図—2.2　プログラムの基本となる構成要素

⑤　計算結果の出力

計算結果をファイルに出力する。

⑥　プログラムの終了

プログラムの終了を表す記述である。

2.4　プログラム例と基本文法

プログラムの構造やプログラミング言語の基本文法について説明するため，**図—2.2**に示したプログラムの基本構成要素が含まれており，本講座でも実際に用いる数値解析法であるニュートン・ラプソン（Newton-Raphson）法[注6]を一例として説明する。

2.4.1　ニュートン・ラプソン法（NR法）

NR法は非線形方程式を解く数値解析手法の一つである。非線形方程式とは未知数である変数の次数が1次ではない方程式であり，後の例で示す指数関数を含む方程式も非線形方程式である。地盤の解析においては，土の有効応力とひずみの関係が線形ではなく，例えばe-$\log p'$関係にみられるように非線形となることからNR法は必須となる。また，1変数の非線形方程式だけでなく，他変数の連立非線形方程式も解くことができる。例えば，6章の圧密解析では土骨格の変位と間隙水圧の二つの変数を持つ連立非線形方程式が支配方程式となっており，ここでも2変数に対するNR法が用いられる。

説明のための例題として，次の非線形方程式をNR法で数値的に解く手順を説明する。

$$\exp(x)-2=0 \quad \cdots\cdots(2.1)$$

ここに，xが変数であり，正解は$x=\ln 2=0.693...$である。ここで，左辺を以下のようにおく。

$$y=\exp(x)-2 \quad \cdots\cdots(2.2)$$

式(2.1)を解くということは，$y=0$となるときのxを求めることである。**図—2.3**をもとに以下にその手順を示す。

①　xの初期値を設定する。ここでは，

$$x_{(1)}=0.000 \quad \cdots\cdots(2.3)$$

とする。ここに，$x_{(1)}$は1回目の繰返し計算におけるx

注5)　かつ効率的である必要があるが本講座では扱わない。

注6)　万有引力のIsaac Newton（1711, 1736発表）とJoseph Raphson（1690発表）がそれぞれ提案した。単にNewton法と呼ばれる場合もある。

の値である。

② 現在のxの値における関数yの値とこの点における接線勾配y'を求める。1回目の繰返し計算では，**図—2.3(a)**に示す接点Aで，

$$y_{(1)}=-1.000 \quad y'_{(1)}=\frac{dy}{dx_{(1)}}=1.000 \quad \cdots\cdots(2.4)$$

となる。

③ この接点Aにおける接線と$y=0$との交点を求める。1回目の繰返し計算では**図—2.3(a)**に示すように，接点Aとx軸との交点aが得られ，関係式は次式で与えられる。

$$\frac{y_{(1)}-0.000}{x_{(1)}-x}=y'_{(1)} \quad \frac{-1.000-0.000}{0.000-x}=1.000 \cdots\cdots(2.5)$$

これより，交点aでのxが以下のように得られる。

$$x=x_{(1)}-\frac{y_{(1)}}{y'_{(1)}} \quad x=0.000-\frac{-1.000}{1.000}=1.000 \quad \cdots\cdots(2.6)$$

④ xの変化量がその時点でのxの値に対して十分に小さくなれば，繰返し計算を終了する。あるいは，yが十分にゼロに近くなれば，繰返し計算を終了する。例えば，次式によってxの変化の大きさを計算する。

$$\text{error}=\left|\frac{x-x_{(1)}}{x}\right|=\left|\frac{1.000-0.000}{1.000}\right|=1.000 \quad \cdots\cdots(2.7)$$

ここに，errorは誤差であり，1回目の繰返し計算での誤差は1.000である。この値が数値解析上「十分に小さい」ことを判断するには，誤差がある許容値より小さいことを判断する。例えば，許容値を1.0e−5（1.0×10^{-5}）とすれば，1回目の繰返し計算における誤差は許容値より大きくなる。この場合，この交点でのxの値を2回目の繰返し計算における新しいxの値とし，

$$x_{(2)}=x=1.000 \quad \cdots\cdots(2.8)$$

再び②〜④を繰り返す。

2回目の繰返し計算では**図—2.3(b)**に示すように，

② $y_{(2)}=0.718 \quad y'_{(2)}=\frac{dy}{dx_{(2)}}=2.718 \cdots\cdots(2.9)$

③ $x=x_{(2)}-\frac{y_{(2)}}{y'_{(2)}} \quad x=1.000-\frac{0.718}{2.718}=0.736 \cdots\cdots(2.10)$

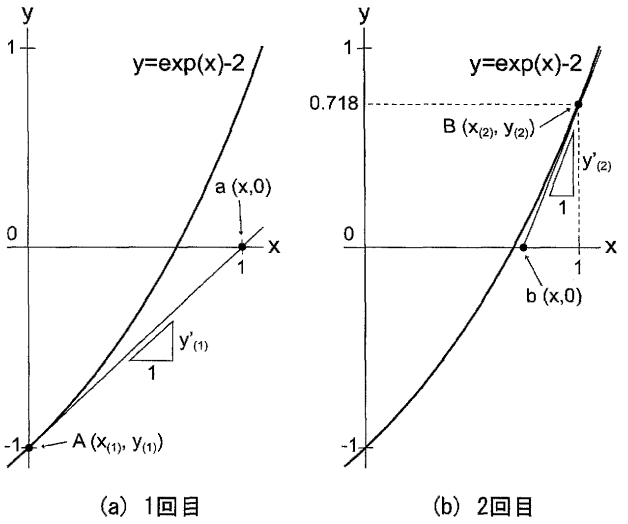
④ $\text{error}=\left|\frac{x-x_{(2)}}{x}\right|=\left|\frac{0.736-1.000}{0.736}\right|=0.359 \cdots(2.11)$

となり，繰返し回数2回で正解に近づき，誤差もずいぶん小さくなることがわかる。ただし，誤差の許容値1.0e−5に対してはまだ大きいので3回目の繰返し計算に進む。この条件では後のプログラムによる実行例で示すように，5回目の繰返し計算によって誤差が許容値以下となる（この状態を計算が収束したと呼ぶ）。

以上の式をより一般的に表すと，繰返し回数k回における解$x_{(k+1)}$（繰返し回数$k+1$回の初期値）は次式のように表せる。

$$y'_{(k)}\Delta x_{(k)}=-y_{(k)}, \quad \Delta x_{(k)}=-\frac{y_{(k)}}{y'_{(k)}}, \quad x_{(k+1)}=x_{(k)}+\Delta x_{(k)} \quad \cdots\cdots(2.12)$$

ここに，$x_{(k)}$は繰返し回数k回におけるのx値，$\Delta x_{(k)}$は繰返し回数k回におけるxの補正量，$y_{(k)}$は繰返し回数k回における解くべき方程式の値，$y'_{(k)}$は$x_{(k)}$での接線勾配である。式(2.12)の第1式は接点$(x_{(k)}, y_{(k)})$における接線の方程式であり，非線形方程式$y=0$の線形化方程式と呼ぶ。また，線形化方程式の未知数$\Delta x_{(k)}$にかかる$y'_{(k)}$は接線係数，線形化方程式の右辺である$y_{(k)}$は残差と呼ぶ。ここで，NR法に関するいくつかの特徴を述べる。



図—2.3 NR法の図解

(1) 速い収束

ここでは繰返し回数5回で所定の精度を達成することができた。後の実際のプログラムによる収束過程をみるとわかるように，誤差の負の指数部の値がおよそ2倍ずつ大きくなっており（例えば，1.0e−3→1.0e−6→1.0e−12），現在の繰返し計算時の誤差の値は一つ前の繰返し計算時の誤差の値のおよそ二乗となる。このような収束過程は二次収束と呼ばれ，繰返し回数が増加するにつれ収束が加速し，比較的少ない繰返し回数で収束が得られる。一般にNR法ではこの二次収束が得られ，収束が速い点が長所である。

(2) 初期値の設定の問題

ここでは手順①でxの初期値として0.0を仮定した。NR法では初期値における接線勾配y'を用いて，解を求めていくことから，その数値解は初期値に依存する結果となる。この例のように解が一つの場合は良いが，解が二つ以上ある場合には解が初期値によって異なる場合もある[注7)]。

(3) 接線係数がゼロとなる問題

手順③の式(2.6)で分かるように接点での接線勾配y'がゼロだと，いわゆる「ゼロ割」となり商が計算機の扱うことができる数値の範囲を超えてしまう（オーバーフローと呼ばれる）ため，例えば"NaN"（Not a Number）などと記憶される。プログラミングではこの

注7) 一般に非線形解析では増分解析（一度にすべての荷重を与えず，荷重を仮想的に分割して，小さい荷重増分に対する解析を繰り返し，徐々に荷重を与える）を用いることが多い。このため，NR法の初期値と求める数値解に大きな差はないことから，よほど非線形性が強くない限り，初期値の設定が問題となることは少ない。

ようなゼロ割が発生しないように注意する必要がある。後のプログラム例ではゼロ割を回避することに配慮する。

2.4.2 NR 法の Fortran によるプログラムと基本文法

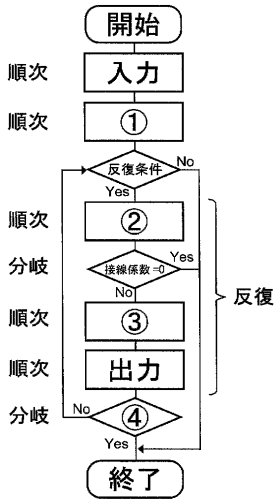
NR 法の Fortran によるソースコードの一例を示し，Fortran の基本的な文法といくつかの構文について説明する。また，2.4.1の例の実行結果についても説明する。

(1) NR 法のアルゴリズムとフローチャート

2.4.1で示した①～④の計算手順から，プログラムの基本の三要素（順次，反復，分岐）を用いた数値計算上の計算手順（アルゴリズム）を作成する。図—2.4に①～④からなるアルゴリズムのフローチャートと三要素との対応関係を示す。NR法は手順も少なく，比較的簡単なアルゴリズムではあるが，基本の三要素はいずれも入っている。このような簡単なアルゴリズムであれば，図のようなフローチャートを描く必要もないが，より複雑なアルゴリズムの場合，フローチャートを描くことはアルゴリズムの確認と整理に役立つものとなる。

(2) ソースコード例

Fortran によるソースコードの例とその説明文を以下に示す。



図—2.4　NR 法のアルゴリズムとフローチャート

リスト—2.1　NR 法の Fortran によるソースコード

行	ソース コード
1	PROGRAM nr
2	IMPLICIT NONE
3	INTEGER,PARAMETER::p=SELECTED_REAL_KIND(15)
4	REAL(p),PARAMETER::zero=1.0e-15
5	INTEGER(4):: i,imax
6	REAL(p):: a,b,x,y,dx,dy,er,er_tol
7	! パラメータの設定
8	imax=10; er_tol=1.0e-5
9	OPEN(1, FILE="input.txt")
10	OPEN(2, FILE="output.txt")
11	READ(1,*) a, b, x
12	WRITE(2,'(3(A,E12.3))') & "a: ",a,", b: ",b,", x: ",x
13	DO i=1,imax
14	y=a*EXP(x)+b; dy=a*EXP(x)
15	IF(ABS(dy) > zero) THEN
16	dx=-y/dy; x=x+dx
17	ELSE
18	WRITE(2,'(A)') "dy がゼロ"; STOP
19	END IF
20	er=ABS(dx/x)
21	WRITE(2,'(I2,1P,5E12.3)') i,y,dy,dx,x,er
22	IF(er < er_tol) EXIT
23	END DO
24	END PROGRAM nr

行	説　明　文
1	プログラム開始。nr は任意のプログラム名。Fortran は大文字・小文字の区別はない。ここでは PROGRAM など文法上定められた用語は大文字で書く。
2	用いる変数は必ず定義する。変数の入力ミスを防ぐためここでは他言語同様に全ての変数を定義する。2 行目以降，2 文字インデントしているのはプログラムを見やすくするためで実行には影響しない。
3	実数型変数の有効桁数（この例では15桁）を指定するための整数型パラメータ p を定義する。実数に“_p”を付加することで p で指定した桁数が有効となる。PARAMETER は計算の途中で値が変わらない変数として用いられる。
4	計算上のゼロを表すパラメータ zero を定義し，値を代入する。
5	整数型変数（4 バイト）の定義。
6	実数型変数（桁指定）の定義。
7	! は注釈（コメント）でプログラムとは無関係に自由に記述できる。行の最後にも追加できる。日本語も可。
8	最大繰返し回数 imax の設定。変数 imax に10が代入される。続けて，許容誤差 er_tol の設定。変数 er_tol に 1.0e-5 が代入される。Fortran では改行が順次実行の上での行の区切りを表すが，このようにセミコロン ; を用いて行の区切りを表すこともできる。
9	用いるファイルを指定する OPEN 文。ファイル番号 1 にファイル input.txt を割り当てる。
10	ファイル番号 2 にファイル output.txt を割り当てる。
11	ファイル番号 1 より計算条件の入力。a，b は解くべき方程式の係数で，先の例では a=1.0，b=-2.0である。x は解の初期値で，先の例では x=0.0である。“*”は自由書式を示しており，データはスペース，タブ，カンマなどで区切る。
12	ファイル番号 2 に入力した a，b，x の出力。'()'で囲まれた部分は書式を表している。A は文字，E12.3は小数点以下 3 桁で，指数部を含めて12桁の指数形式の実数である。行末の & は行の継続を示す。
13	DO 文による反復開始。i は 1 から，1 ずつ増加し，最大 imax。23行目までを i=imax となるまで反復。
14	現在の残差 y および現在の接線係数 dy の計算。*は掛け算である。
15	分岐の開始。接線係数 dy の絶対値 ABS(dy) が zero より大きい場合，16行を実行。それ以外の場合，18行を実行。ABS は絶対値を計算する組み込み関数。
16	現在の x の補正量 dx の計算および新しい x の更新。/は割り算である。
17	接線係数 dy の絶対値 ABS(dy) が zero 以下の場合
18	ファイル番号 2 に "dy がゼロ" と出力して，STOP 文によりプログラムを停止。

19	分岐の終了。
20	収束判定用誤差 er の計算。
21	ファイル番号 2 に結果の出力。'()'で囲まれた部分は書式を表している。I2 は 2 桁の整数。1P は実数表示部の小数点を右へ 1 桁ずらす。5E12.3は小数点以下 3 桁で，指数部を含めて12桁の指数形式の実数（E12.3）を 5 つ並べる。
22	収束判定のための分岐。誤差 er が許容値 er_tol より小さいなら EXIT で DO 文を抜け，反復を終了する。それ以外なら次の行へ。
23	反復終了。13行目に戻り，i が 1 増加する。
24	プログラム終了

(3)　基本文法

ここでは本講座を理解するために最低限必要な文法について説明[注8]する。なお，Fortran の文法の詳細については専門書[1]やインターネット[注9]を参照されたい。

(a)　変数の定義

本講座で扱う変数は，主に整数と有効桁指定の実数である。これらは INTEGER (4) 文と REAL (p) 文を用いて 3～6 行目で示したように定義する。INTEGER (4) は 4 バイトの整数を表しており，-2^{31} から $2^{31}-1$ までの値を扱うことができる。REAL (p) は10進数での p 桁指定の実数であり，この例では15桁を指定しており，REAL (8)（8 バイトの実数，倍精度型実数，10進数で約15桁）と同等の精度を有する。整数や実数などは変数の型と呼ばれる。この他，文字型の変数（CHARACTER 文で指定）も一部で用いる。

(b)　変数への数値の代入

プログラム中の等号“＝”は数学での等号とは異なり，左辺の変数に右辺の値（数値，変数，計算式）を代入することを意味している。8, 14, 16, 20行はすべて代入文である。

(c)　反復の構文

様々な反復のための構文があるが，ここでは DO 文を用いている。13～23行目が DO 文による反復を示しているが，改めて以下の例で説明する。

```
x=0.0
DO i=1, 10
  x=x+1.0
END DO
```

この反復は x に1.0を加える計算を10回行うものであり，この結果，x＝10.0となる。i＝1, 10は i を 1 から10まで 1 ずつ増加させることを意味している。すなわち，1 が i の初期値，10が i の終値である。参考までに，i＝1, 10, 2 とすると，i＝1, 3, 5, 7, 9 と変化し，i を 2 ずつ増加させることができる。つまり，i＝1, 10は i＝1, 10, 1 と同じ意味であり，最後の 1 は i の増分値である。なお，DO から END DO までの反復処理の部分が一段下がっている（インデント，スペースあるいは tab キーを用いる）のはプログラムを見やすくするためであり，実行には影響しない。

22行に示すように，DO 文の途中で反復を終了する場合には EXIT 文を用いる。EXIT 文が実行されると，反復の途中でも反復が終了し，END DO の次の処理がなされる。例えば，

```
x=0.0
DO i=1,10
  IF (i==5) EXIT
  x=x+1.0
END DO
```

の場合，x＝4.0となる。また，DO 文の途中で CYCLE 文が実行されると，それ以降の処理がスキップされ END DO に移動し，次の i の計算を行う。例えば，

```
x=0.0
DO i=1,10
  IF (i==5) CYCLE
  x=x+1.0
END DO
```

の場合，x＝9.0となる。この他，反復と分岐を組み合わせた DO WHILE 文などもある。

(d)　分岐の構文

22行目の IF 文は分岐の最も単純な構文である。括弧内が真なら右側の処理を実行し，偽なら次の行に移る。また，15～19行目の IF THEN 文は分岐による処理の違いを表すためのものである。改めて以下の例で説明する。

```
i=1; x=0.0
IF (i==1) THEN
  x=1.0
ELSE
  x=2.0
END IF
```

この IF THEN 文では，括弧内が真なら THEN から ELSE の間の処理を実行し，括弧内が偽なら ELSE から END IF の間の処理を実行する。この場合，x＝1.0となる。括弧内は論理式と呼ばれ，以下の記号を用いて記述する。

大きい	>
大きいか等しい	>=
小さい	<
小さいか等しい	<=
等しい	==
等しくない	/=

例えば，

```
i=1; x=0.0
IF(i/=1) THEN
  x=1.0
ELSE
  x=2.0
END IF
```

の場合，x＝2.0となる。さらに分岐の数を増やす場合は

[注8] 基本は前述の順次・反復・分岐であり，ここに示す 2 ページ足らずであるので臆する必要はない。

[注9] 例えば，http://ace.phys.h.kyoto-u.ac.jp/～tomita/education/fortran90/sec0.html

次のようにELSEIF文を用いることができる。

```
i=2; x=0.0
IF(i==1) THEN
  x=1.0
ELSEIF(i==2) THEN
  x=2.0
ELSE
  x=3.0
END IF
```

この場合，x＝2.0となる。この他，より分岐が多い場合にはSELECT CASE文も利用できる。

(e)　サブルーチン

この例のように簡単なプログラムでは一つのプログラムで記述可能であるが，有限要素法のように様々な計算結果を組み合わせて最終的な答えを出すプログラムではサブルーチンと呼ばれるサブプログラムを用いる方がわかりやすい。以下はサブルーチンを利用した例である。

リスト―2.2　サブルーチンのソースコード例

行	ソ　ー　ス　コ　ー　ド
1	PROGRAM sub
2	IMPLICIT NONE
3	REAL(8):: x
4	x=2.0
5	CALL cal_power(x)
6	WRITE(*,'(A,F10.3)') "x=", x
7	END PROGRAM sub
8	!
9	SUBROUTINE cal_power(a)
10	REAL(8):: a
11	a=a*a
12	END SUBROUTINE cal_power

行	説　　明　　文
1	メインプログラムsubの開始。
3	倍精度実数xの定義。
4	xに2.0を代入。
5	サブルーチンcal_powerの呼び出し。xの値がサブルーチンに引き渡される。サブルーチンの処理が終了した際には，逆にサブルーチン内で計算された値がxに代入される。
6	xの出力。x=4.0となる。
7	メインプログラムsubの終了。
9	サブルーチンcal_powerの開始。括弧内のaが引数として，メインプログラムのxに対応する。
10	倍精度実数aの定義。
11	aの二乗をaに代入。
12	サブルーチンcal_powerの終了。aの値がメインプログラムのxに代入される。

PROGRAMからEND PROGRAMまでがメインプログラム（sub），SUBROUTINTEからEND SUBROUTINEまでがサブルーチン（cal_power）であり，それぞれプログラム単位と呼ばれる。サブルーチンを呼び出す場合は，5行目のように，

CALLサブルーチン名（引数）

と記述する。引数とは，呼び出し側のメインプログラムとサブルーチンで値をやり取りする変数であり，ここではxが引数（実引数）となっている。サブルーチンではメインプログラムから渡されたxを，異なる名前のaを引数（仮引数）として受け取っている。aのようにサブルーチン内部のみで有効な変数をローカル変数[注10]と呼ぶ。ローカル変数はメインプログラムや他のサブルーチンとは独立に定義できるため，この例のように異なる名前でも良い（もちろん同じ名前でもよい）が，xとaの変数の型（この例では8バイトの実数）は同一でなければならない。

(f)　配列

この例では変数が一つの値を記録できるだけであるが，ベクトルやマトリックスを扱うには一つの変数名で複数の値を記憶できる配列を使うと便利である。以下は配列を利用した例である。Fortranでは，配列内の要素をまとめて操作できること，組込みの配列関数を利用できることなど配列の扱いが容易である[注11]。

リスト―2.3　配列のソースコード例

行	ソ　ー　ス　コ　ー　ド
1	PROGRAM array
2	IMPLICIT NONE
3	INTEGER(4):: sca, vec1(1:2), vec2(1:2), & vec3(1:2), mat(1:2,1:2)
4	sca=0; vec1=0; vec2=0; vec3=0; mat=0
5	vec1(1)=2; vec1(2)=1
6	vec2(1)=1; vec2(2)=3
7	mat(1:2,1)=vec1(1:2)
8	mat(1:2,2)=vec2(1:2)
9	sca=DOT_PRODUCT(vec1(1:2),vec2(1:2))
10	vec3(1:2)=MATMUL(mat(1:2,1:2),vec1(1:2))
11	WRITE(*,'(A,I5)') "sca=", sca
12	WRITE(*,'(A,2I5)') "vec3=", vec3(1:2)
13	END PROGRAM array

行	説　　明　　文
3	vec1，vec2，vec3は一次元の配列である。括弧内の(1:2)は添え字であり，配列の下限が1，上限が2であることを示している。これらは2個の配列要素を持っており，2個の数値を記憶できる。下限が1なら省略してvec1(2)でもよいが，以下の配列同士の演算では下限も指定し演算範囲を明確にする。 matは二次元の配列である。2×2=4個の配列要素を持っており，4個の数値を記憶できる。

注10)　これに対して，メインプログラムやサブルーチンで引数を用いることなく共通に使える変数をグローバル変数と呼ぶ。後の有限要素法のプログラムでは，この例のように引数でメインプログラムとサブルーチン間の変数のやり取りをすると引数が多くなるので，モジュールを用いてグローバル変数を定義する。モジュールとは，グローバル変数，関数やサブルーチンなどプログラム全体で共通に用いるものを記述するプログラム単位である。

注11)　二次元以上の解析でベクトルやテンソルを配列として扱えば，これらの配列演算によってプログラムを簡潔に記述することができる。

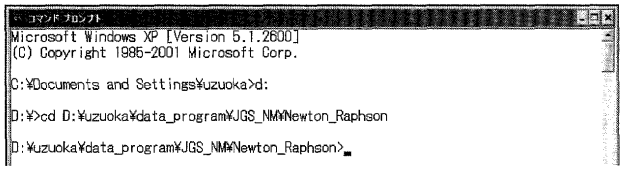
4	配列の初期化。配列は必ず初期化して用いる。
5	vec1 を (2, 1) に設定。
6	vec2 を (1, 3) に設定。
7	mat の 1 列目に vec1 を設定。このように配列の添え字を対応させて，まとめて代入することができる。
8	mat の 2 列目に vec2 を設定。
9	vec1 と vec2 の内積の計算値を sca に代入する。DOT_PRODUCT は二つの引数（一次元配列）の内積を計算する組み込み関数である。二つの一次元配列の添え字の大きさは同一である必要がある。
10	mat と vec1 の積を vec3 に代入する。MATMUL は二つの引数（二次元配列と一次元配列，二次元配列同士）の積を計算する組み込み関数である。対応する添え字の大きさは同一である必要がある。
11	sca=5 となる。
12	vec3=(5, 5) となる。

(4)　プログラムの実行例

リスト2.1で示したソースプログラムを用いて計算する手順は以下のとおりである。

①　ソースファイルを作成する。あるフォルダにおいてテキストエディタで先のソースを記述したファイルをファイル名 nr.f90 として保存する。

②　コマンドプロンプトを起動し，ソースファイルのあるフォルダに移動する。



③　コマンドプロンプトにおいて以下のように入力し，ソースファイル（nr.f90）のコンパイルを行う。

```
D:¥uzuoka¥data_program¥JGS_NM¥Newton_Raphson>g95 -o nr.exe nr.f90
```

正常にコンパイルできれば，同じフォルダに nr.exe ファイルが作成される。この nr.exe がプログラムの実行形式のファイルである。コマンド中の“-o nr.exe”は nr.exe という名前の実行形式のファイルを出力させるコンパイル時のオプション[注12)]である。このようにコンパイルとはソースファイルを実行可能なファイルに変換する作業である。

④　入力用のファイル input.txt を準備する。テキストエディタで以下の1行を入力したファイルを nr.exe があるフォルダに保存する。

```
1.0  −2.0  0.0
```

この例はスペース区切りであるが，タブあるいはカンマ区切りでもよい。

⑤　最後に，コマンドプロンプトにおいて次のように入力し，プログラムを実行する。

```
D:¥uzuoka¥data_program¥JGS_NM¥Newton_Raphson>nr.exe
```

nr.exe と同じフォルダに新しく output.txt が作成され，以下のように出力されていれば，正常終了である。

```
a:  0.100E+01, b:  −0.200E+01, x:  0.000E+00
 1 1.000E+00  1.000E+00   1.000E+00 1.000E+00 1.000E+00
 2 7.183E−01  2.718E+00  −2.642E−01 7.358E−01 3.591E−01
 3 8.707E−02  2.087E+00  −4.172E−02 6.940E−01 6.011E−02
 4 1.791E−03  2.002E+00  −8.947E−04 6.931E−01 1.291E−03
 5 8.010E−07  2.000E+00  −4.005E−07 6.931E−01 5.778E−07
```

1行目は input.txt より入力した値であり，プログラムの12行目の WRITE 文による出力である。2行目以降がプログラムの21行目の WRITE 文による出力である。1列目の整数が繰返し回数 i を示しており，繰返し回数5回で計算が終了している。このときの誤差 er（6列目）は許容値 1.0e−5 より小さくなっている。また，残差 y の値（2列目）も小さい値となっており，$y=0$ がほぼ満足され，x の値（5列目）はほぼ正解に近い値となっている。

参　考　文　献

1)　例えば，牛島　省：数値計算のための Fortran90/95 プログラミング入門，森北出版，2007.

付録：

NR 法の C および Java によるプログラム例と基本文法について，地盤工学会のホームページを通じて公開している。

注12)　コンパイル時のオプションの詳細は g95 の readme.txt やマニュアルを参照されたい。