2005 年度情報処理 II　第 11 回

数学のためのコンピューター (3) 方程式の数値解法

桂田 祐史（かつらだ まさし）

2005 年 7 月 7 日

ホームページは http://www.math.meiji.ac.jp/~mk/syori2/

1　数学とコンピューター

数学のためのコンピューターは NSG の三本柱

1. Numerical (computation) ... 数値計算
2. Symbolic (computation) ... 数式処理
3. Graphics (or visualization) ... グラフィックス (可視化)

今日は N, すなわち数値計算、特に方程式の数値解法について考える。

コンピューターによる方程式の数値解法は「繰り返し」がキー

1. 線型方程式 (1 次方程式) でも、未知関数の個数 N が非常に大きい問題が現われる
   例えば N=1 億 (原子炉圧力容器の構造計算であるとか[1])
2. 非線型方程式の場合、問題の自由度が低くても反復計算になる。
   (a) 有限回の四則では exact (厳密) に解けない問題がほとんどである。
   (b) 「反復法」で多くの問題が解ける。
   すなわち、真の解 x^* をある列 $\{x_n\}$ の極限としてとらえ ($\lim_{n\to\infty} x_n = x^*$)、十分大きな n に対して x_n を x^* の近似として採用する。近似解ではあるが、コンピューターで数値計算する限り、有限精度であることは避けられないので、exact な方法 (もしそれがあったとして) とほとんど差がない[2]。

[1] 本来は、偏微分方程式で、未知数の個数は無限というべき問題だが、適当な離散化により、有限次元の問題になる。

[2] 実際、連立 1 次方程式は Gauss の消去法など exact な計算法が知られているが、CG 法などの反復法が採用されることも多い。

例 **1.1** $x^3 - 4x^2 + 3x + 4 = 0$ の実数解を求めよ、という問題の解は

$$x = \frac{1}{3}\left(4 - \frac{7}{\sqrt[3]{44 - 3\sqrt{177}}} - \sqrt[3]{44 - 3\sqrt{177}}\right)$$

と求まるが、実際的必要から数値を求めたい場合 $\sqrt{\ }$, $\sqrt[3]{\ }$ の値を数値計算する必要がある。最初から二分法や Newton 法で直接解を計算する方が簡単である可能性が高い。 ■

2 C を使おう

2.1 Why C?

現時点で万能といえるプログラミングは存在しない。
それでなぜ C 言語によるプログラミングを学ぶのか？

1. まあまあのバランスで「使える」言語である。
   C はかなりコンピューター寄り[3] (原始的？) だが、そこそこ人間にもやさしい (登場した頃は高級言語と言われた)。
   伝統的には「数値計算は FORTRAN (がよい)」と言われていた。C は数値計算もまあまあこなしす[4]、古い FORTRAN が苦手なことの多く[5]をそつなくこなすことが多い。

2. 他のプログラミング言語を学ぶときに、C を学んだことは役に立つ。

   - もともとプログラミングの本質は言語が変ってもそうそう変らない。
   - 特に C が現在の実際のコンピューターよりであることはポイント。
   - Java, C++ など C 風[6]の有力な言語がある。

2.2 How C? (どうやるのか、復習)

Linux のような UNIX 互換 OS でどのように C でプログラミングするか、復習しておこう。

テキスト・エディター (emacs など) でソース・プログラム (ファイル名の末尾は .c) を作成し、C コンパイラー (gcc など) で、実行形式への翻訳 (コンパイル) をして、できた実行形式を実行する。

ソース・プログラムを作成し、コンパイル&リンク、実行

```
a308-01% emacs prog.c &
a308-01% gcc -o prog prog.c -lm
a308-01% ./prog
```

[3] 2 進法, ポインター (メモリーのアドレス) など、生のコンピューターを意識させてしまうというのは、到底抽象度が高いとは言いかねる。

[4] いざとなれば、FORTRAN で書かれた数値計算ライブラリィとリンクすることもできる。

[5] OS などのシステムの記述、少し前の時代のワードプロセッサー, ヴィデオ・ゲームなどのソフトウェア、家電製品の組み込み用マイコンのためのプログラムなど。

[6] C++ はほぼ上位互換性を持っている (C のプログラムは C++ のプログラムでもある)。Java もプログラムの体裁では C のプログラムに似ているところが多く、多少は親近感が抱けるはず。

注意点

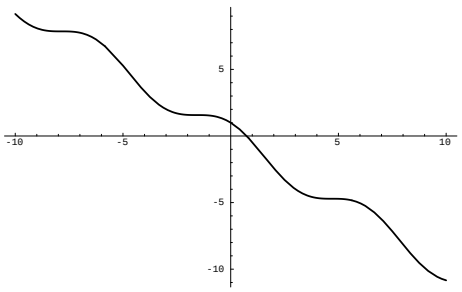
- `-o prog` は実行形式の名前を `prog` にせよ、という指示である。これを省くと `a.out` という名前の実行形式ができる (当然実行は `./a.out` となる)。
- `-lm` (`l` は `L` の小文字) は、数学関数ライブラリィ (libm.a) をリンクせよ、という指示である。

3 反復法による方程式の解法

3.1 例題

「方程式 $\cos x - x = 0$ の実数解 (の数値) を精度良く求めよ。」

$f(x) = \cos x - x$ とおいて、f を少し調べてみれば (このすぐ後に述べる)、この方程式にはただ一つの実数解があって、それは区間 $(0, 1)$ にあることが分かる。大雑把に言うと、グラフの概形は $y = -x$ をサイン・カーブ風に波打たせたものになる。この唯一の解を求めることが目標である。



方程式が区間 $(0, 1)$ にただ一つの解を持つことの証明

まず $f'(x) = -\sin x - 1 \leq 0$ $(x \in \mathbf{R})$ で、特に $x = \pi/2 + 2n\pi$ $(n \in \mathbf{Z})$ 以外のところでは $f'(x) < 0$ であるから、f は狭義の単調減少関数である。そして $f(0) = 1 > 0$, $f(1) = \cos 1 - 1 < 0$ ゆえ、中間値の定理によって、方程式 $f(x) = 0$ は区間 $(0, 1)$ 内に少なくとも一つの解を持つが、f の単調性からそれは $\mathbf{R}$ 全体でただ一つの解であることが分かる。■

3.2 レポート課題 8

時間がないので、やるべきことは細かく指示します。質問はなるべく今日中にどんどんして下さい (次回 Windows になる可能性が高いので)。

Subject は「情報処理 II 課題 8」、締め切りは 7 月 16 日、プログラムとその実行結果、その説明の 3 点セットが必要[7]。

(1) 与えられた正数 a に対して $\sqrt{a}$ を計算するプログラムを作り、$\sqrt{2}, \sqrt{3}, \sqrt{5}$ を計算し、ライブラリィ関数 `sqrt()` の結果と比較せよ。

[7]もう少し詳しく言うと、C のプログラムと、それを実行するときどういう入力を与えて結果がどうであったかのデータ (ここまでで実験の追試が可能になる)、その結果をどう判断するか自分なりの説明 (必ずしも長い必要はない)。

(2) 与えられた $a \in [-1, 1]$ に対して (普通の三角関数は利用してもよいが、逆三角関数は利用せずに) $\arcsin a$ を計算するプログラムを作り、$\arcsin \frac{1}{2}$ を計算し、ライブラリィ関数 `asin()` の結果と比較せよ。

ヒント

- 後で紹介するサンプル・プログラムを書き直すという手順で作成できる。例えば

```
a308-01% cp bisection.c kadai8-1.c
a308-01% emacs kadai8-1.c &
```

  というようにする。

- $\sqrt{a}$ は例えば $x^2 - a = 0$ の正の解と解釈できる。
- $\arcsin a$ は例えば $\sin x - a = 0$ の $[-\pi/2, \pi/2]$ 内の解と解釈できる。
- (老婆心) C 言語のプログラム中に `1/2` という式を書くと値は 0 であり、$\frac{1}{2}$ にはならない。`1.0/2.0` あるいは `0.5` とする。

3.3 二分法

まずサンプル・プログラム bisection.c[8] を示す。ブラウザーで読み込み、[ファイル] メニューから [名前をつけて保存] を選び、保存すること。

コンパイルと実行

```
a308-01% gcc -o bisection bisection.c
a308-01% ./bisection
探す区間の左端，右端，要求精度:  0 1 1e-15
... (実行結果は実際に見てもらうことにします)
```

計算の原理は中間値の定理「連続な関数 $f\colon [a, b] \to \mathbf{R}$ が、$f(a)f(b) < 0$ を満たせば、(a, b) に解が少なくとも一つ存在する」とその区間縮小法による証明 (付録に書いておいた) に基づく。

$[a, b]$ に解が存在するならば、二つに分割した区間 $[a, (a+b)/2]$, $[(a+b)/2, b]$ のどちらかに存在する (両方に存在することもある) が、どちらであるか判断できれば、繰り返すことで区間の幅を半分半分にしていけて、解を追い詰めることができる。

なお、上の計算では要求精度 (区間の幅がどこまで小さくなったら反復を停止するか) を `1e-15` (意味は 1×10^{-15} という意味) としたが、これは演習に用いている C 言語処理系の double の精度が 10 進法に換算して 16 桁弱であることから決めたものである。

(詳しいことはこの文書の付録を参照せよ。)

[8] http://www.math.meiji.ac.jp/~mk/syori2-2005/bisection.c

3.4 Newton 法

例年 Newton 法を解説し、課題も出しているのだが、今年度は時間の関係でカットする。なお、以下に付録として、昨年度の資料を添付しておくので、興味があれば目を通して見て下さい。

A 数学とコンピューターの関係

A.1 ある講演会から

つたない説明だが、2002 年度日本数学会の市民後援会のときの資料

『コンピューターは数学の望遠鏡』
http://www.math.meiji.ac.jp/~mk/shiminkouen/

を紹介しておく。

どういう内容か

- 数学とコンピューターは (小さく見積もっても) 年齢比 40 以上、関係はイマイチである
- 次のように色々な視点がありうる。
  1. コンピューターのための数学 (論理数学、計算可能性の理論など[a])
  2. 数学の応用とコンピューター (鬼に鉄棒(かなぼう)、数学にコンピューター)
  3. 数学のためのコンピューター

  主に 3 番目の観点から論じている。

[a] どうも、工学系の人は新しもの好きで、一度出来てしまったことには冷淡なことが多いようだが、これらが根本的に重要であることを忘れてはいけない。

A.2 私の主張: 計算を見直そう

今、数学を学んでいる君達には、計算することの意味を考えて勉強をしてもらいたい。世間では「算数・数学とは計算だ！」と考えている人が多い。しかし

単純な計算では分からないこと、計算せずに分かることがある。

例えば、

- $\sqrt{2}$ が無理数であることの証明
- 超越数の存在
- 解の存在証明 (例えば、解の計算はせずに中間値の定理で存在のみ示すなど)

- 解の公式の非存在証明など不可能性の証明

数学科学部学生である君達は、まさに計算の向うの世界を勉強中と言ってよい。
でも、

やり過ぎに注意！計算を軽視しないように

計算もやはり大事であり、特にコンピューターが登場してからは、人手 only の時と色々と事情が異なっている。

もう一度すべてを計算の観点から見直す時期だろう
コンピューターに合った計算法？、それで何が出来るか？

B　数学に役立つ computing

B.1　N, S, G が 3 本柱

数学の研究・学習に役立つ計算 (computing, computation) は、Numerical (computation), Symbolic (computation), Graphics の **NSG が 3 本柱**だと言われる。

Numerical Computation 有限精度の数値計算 (近似計算) によるもの。

- Numerical Simulation は computer の最初の応用であったが、現在でも (数学以外でも) 重要性が高い。
- 解析学の分野の研究で、微分方程式の解の様子などを調べるのに使われる。
- 丸め誤差があるので、数学にとっては「状況証拠に過ぎない」とも言われたが、精度保証付き数値計算も発達してきたので将来は分からない。

Symbolic Computation 数値だけでなく、いわゆる数式の計算を行なうもの。

- 数式処理, computer algebra (計算機代数) などと呼ばれる。
- 丸め誤差がないので、計算結果の信頼性は高い。
- かつては高価なコンピューター環境が必要だったため、普及が遅れたが...

Graphics (特に数値) 計算の結果を分かりやすくするには、可視化が描かせない。

B.2　N, S, G に共通して言えること、比較

- 色々なツールがあるが、今のところ万能ツールはない (専用ツール間の連携は考えられるようになってきている)。
- これらのツール内の計算のアルゴリズムは、従来の数学の「教科書」に載っているもの[9]とはかなり違うことが多い。

[9]それらは手計算で解ける規模の問題向きであって、コンピューターが相手にするような規模の問題には不適当なことが多い。

- C や Fortran などの伝統的なコンパイル形式の手続き型言語では、Symbolic Computation は困難で、Graphics にも使用するコンピューター環境に依存したライブラリィが必要になる[10]。Numerical Computation については (環境から独立した汎用の) 数値計算ライブラリィ[11]が利用できる。

- Numerical Computation については、特定の問題向けのソフトウェア、いわゆる **PSE** (Problem Solving Environment) もあるが、**MATLAB** 等の汎用ツールも重要 (体験・学習・習得の価値がある)。

- Symbolic Computation については、**Mathematica**, **Maple** などの汎用のツール (これらはしばしば Graphics 機能を備えている) もあるが、専門家向けのツールもある (専門家自身が作成したものが多い)。

C Numerical Computation の例 — 方程式を解く

コンピューターで数値計算をして (有限次元の) 方程式を解く方法について学ぶ[12]。厳密解を求めることにすると、線形方程式以外は例外的な状況をのぞいて解けない[13]。しかし、有限精度の解 (近似解) で満足することにすれば、かなり多くの方程式が解けることになる。

ここでは二分法と **Newton** (ニュートン) 法を取り上げるが、これらは解析学の学習とも関係が深い。二分法は中間値の定理の区間縮小法による証明 (中間値の定理はいわゆる「存在定理」であるが、この証明は「構成的な」証明であると言える) そのものであると考えられよう。また Newton 法は陰関数の定理や逆関数の定理の証明に用いることもできるし、実際に陰関数・逆関数の計算に利用できる。

次の例題を考える。

例題 **1:** 二分法によって、方程式 $\cos x - x = 0$ の解を計算せよ。

例題 **2:** Newton 法によって、方程式 $\cos x - x = 0$ の解を計算せよ。

「方程式を解け」という問題はしばしば現れる。基本的で大事な方程式はその性質を学んできたが、それ以外にも色々な方程式がある。方程式は解が存在しても、紙と鉛筆の計算で具体的に解くのが難しいことがしばしばある[14]。この例題の方程式もそういうものの一つで、解がただ一つあることは簡単に分かるが (後の補足を参照)、その解を簡単な式変形等で求めることは出来そうにない。これにコンピューターを用いて取り組もう、ということである。

コンピューターで方程式を扱う場合には、コンピューターならではのやり方がある。有限回の計算で真の解 (無限精度の解) を求めることをあきらめて、真の解を求めるには無限回の演算が必要だが、有限桁の要求精度を持つ解 (近似解) はそこそこの回数の基本的な演算 (四則

[10]ただし Java 言語などの登場でこのあたりの事情は変わりつつある。

[11]例えば連立 1 次方程式や固有値問題などを解くための Fortran サブルーチンを集めた **LINPACK**, **EISPACK**, **LAPACK** などが草分け。これら数値計算ライブラリィは人類の文化遺産だと言う人もいるくらい、多くの人の知恵と努力の結晶である。

[12]方程式を難しくする「原因」として、非線型性と無限次元性がある。ここでは非線型性を取り上げる。

[13]「例外的な状況」は重要でないと勘違いしないように。解けるような例外的な問題には重要なものも多い。

[14]方程式によっては、手計算では実際的な解法がないものもある、というかそういうものの方が多いわけだが、大学二年次までの段階では、具体的に解ける問題を扱うことの方が多いので、ピンと来ないかもしれない。

や初等関数の計算) で求まるような方法 — 近似解法 — を採用する、というものである。従ってアルゴリズムは、大抵繰り返しのあるものになる。

ここで解説する近似解法は、適用できる範囲はかなり広く、コンピューターを使って計算することになる人は、今後も何度も「お世話になる」はずである。

注意 **C.1** ここで紹介するのは近似計算であり、近似計算は数学的に厳密ではないから「意味がない」と素朴に考える人がいるかもしれない。これについては以下の二つのことを指摘しておこう。

1. 近似計算ではあっても「状況証拠」にはなり、本当がどうなっているのかを想像することはでき、役立つことが多い[15]。
2. 計算結果に厳密で具体的な精度の保証をつけることができ、例えば解の存在なども証明できる場合がある (精度保証数値計算)。

D　方程式の分類

方程式といっても色々なものがあるが、ここでは微分や積分を含まない、有限次元の、「普通の」もの、つまり既知関数 $f : \mathbf{R}^n \supset \Omega \longrightarrow \mathbf{R}^m$ を用いて表される、未知数 x についての方程式

$$f(x) = 0$$

について考える。

D.1　線形方程式 — 比較的簡単

f が x の 1 次式である場合、方程式を線型方程式と呼ぶ。これは、f が適当な行列 $A \in M(m, n; \mathbf{R})$, ベクトル $b \in \mathbf{R}^m$ を用いて $f(x) = Ax - b$ と表されるということで、いわゆる (連立) 1 次方程式

$$Ax = b$$

になる。この場合は有限回の四則演算で解が求まる。(良く知っているように) A が n 次正則行列であった場合は $x = A^{-1}b$. 既に何らかの解法[16]を習ったことがあるはずである。この問題はみかけよりも奥が深く、また非常に応用範囲が広いので、実に精力的に研究されていて、面白い手法も少なくないが、この講義では紹介を見送る。

研究課題 **3-1**　連立 1 次方程式を解くための共役勾配法 (CG method) について調べ、プログラムを書いて実験せよ。

---

[15] ここで引き合いに出すのは、少々こじつけかもしれないが、アルキメデス (BC 287–212, シラクサに生まれ、シラクサに没する) の『方法』に、「ある種の問題は、まず工学的な方法で答が明らかになってしまう。もちろん後で幾何学的に証明を付けなくてはいけないのだけれども、それでも最初から答がわかっているのと、一から考えなくてはならないのとでは雲泥の差がある」— 木村俊一, 『天才数学者はこう解いた、こう生きた』、講談社から引用。つまり 2000 年の歴史のある言い訳。

[16] 掃き出し法 (Jordan の消去法)、Gauss の消去法など。理論的には Cramer の方法 (あまり実用的でない)。

D.2　非線形方程式 — なかなか難しい

$f(x)$ が x の 1 次式では表わせないとき、非線形方程式と呼ばれる。もっとも簡単な非線形方程式は、2 次以上の 1 変数の代数方程式

$$a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = 0 \quad (a_n \neq 0)$$

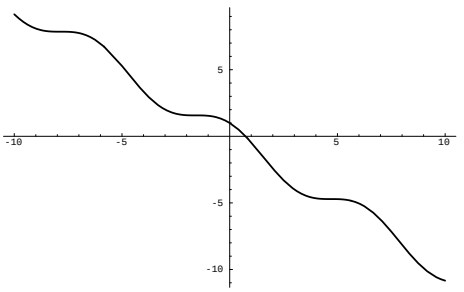
であろう。「n 次代数方程式は n 個の根を持つ」ことは常識として知っているはず。また、次数 n が 2 の場合は「2 次方程式」で、根の公式は中学校で学んでいる (もうすぐ消えた？)。さらに n が $3, 4$ である場合も、(2 次方程式ほどポピュラーではないが) 根の公式[17]がある。ところが、「n が 5 以上の場合は、四則とべき根のみを有限回用いた根の公式は存在しない」ことが **Galois** (ガロア) 理論を用いて証明されている (3 or 4 年の代数学で習うはず)。

代数方程式でもこんな調子なのだから、より一般の非線形方程式を簡単な式変形のみで解くことは、よほど運が良くない限り駄目だ、ということになる。

研究課題 **3-2**　3 次代数方程式を解くための Cardano (カルダノ) の方法について調べ、プログラムを書いて実験せよ。

E　非線形方程式を計算機で解く

ここでは例題の方程式 $\cos x - x = 0$ について考えよう。$f(x) = \cos x - x$ とおいて、f を少し調べてみれば (このすぐ後に述べる)、この方程式にはただ一つの解があって、それは区間 $(0, 1)$ にあることが分かる。大雑把に言うと、グラフの概形は $y = -x$ をサイン・カーブ風に波打たせたものになる。この唯一の解を求めることが目標である。



方程式が区間 $(0, 1)$ にただ一つの解を持つことの証明

まず $f'(x) = -\sin x - 1 \leq 0$ $(x \in \mathbf{R})$ で、特に $x = \pi/2 + 2n\pi$ $(n \in \mathbf{Z})$ 以外のところでは $f'(x) < 0$ であるから、f は狭義の単調減少関数である。そして $f(0) = 1 > 0$, $f(1) = \cos 1 - 1 < 0$ ゆえ、中間値の定理によって、方程式 $f(x) = 0$ は区間 $(0, 1)$ 内に少なくとも一つの解を持つが、f の単調性からそれは $\mathbf{R}$ 全体でただ一つの解であることが分かる。■

E.1　二分法 (bisection method)

微積分で基本的な中間値の定理を復習しよう。

[17] Cardano の方法、Ferrari の方法。もっとも、とても複雑で、紙と鉛筆で計算するのは (少なくとも私は) うんざりしてしまう。

定理 E.1 (中間値の定理) $f:[\alpha,\beta]\to\mathbf{R}$ を連続関数、$f(\alpha)f(\beta)<0$ とすると、$f(c)=0$ となる $c\in(\alpha,\beta)$ が存在する。

(つまり $f(\alpha)f(\beta)<0$ となる α,β があれば、方程式 $f(x)=0$ の解 $x=c$ が区間 (α,β) 内に存在するということ。)

この定理の証明の仕方は色々あるが、代表的なものに区間縮小法を使ったものがある。それは以下のような筋書きで進む。

次の手順で帰納的に数列 $\{a_n\},\{b_n\}$ を定める。

(i) $a_0=\alpha,\ b_0=\beta$ とする。

(ii) 第 n 項 $a_n,\ b_n$ まで定まったとして、$c_n=(a_n+b_n)/2$ とおき、$f(a_n)f(c_n)<0$ なら $a_{n+1}=a_n,\ b_{n+1}=c_n$, そうでないなら $a_{n+1}=c_n,\ b_{n+1}=b_n$ とする。

すると、

$$a_0\le a_1\le a_2\le\cdots\le a_n\le a_{n+1}\le\cdots,\quad \cdots\le b_{n+1}\le b_n\le\cdots\le b_2\le b_1\le b_0$$

$$a_n<b_n\le b_0\quad(n\in\mathbf{N})\quad\text{さらに}\quad a_0\le a_n<b_n\quad(n\in\mathbf{N}),$$
$$b_n-a_n=(\beta-\alpha)/2^n\to 0\quad(\text{as } n\to\infty),$$
$$f(a_n)f(b_n)\le 0\quad(n\in\mathbf{N}).$$

これから

$$\lim_{n\to+\infty}a_n=\lim_{n\to+\infty}b_n=c,\quad \alpha<c<\beta$$

と収束して

$$f(c)=0$$

が成り立つことが分かる。■

以上の証明の手続きから、$f(\alpha)f(\beta)<0$ となる α,β が分かっている場合に、方程式 $f(x)=0$ の近似解を求めるアルゴリズムが得られる (以下では $\leftarrow$ は変数への代入を表す)。

二分法のアルゴリズム

(1) 目標とする誤差 ε を決める。

(2) $a\leftarrow\alpha,\ b\leftarrow\beta$ とする。

(3) $c\leftarrow(b+a)/2$ として $f(a)f(c)<0$ ならば $b\leftarrow c$、そうでなければ $a\leftarrow c$ とする

(4) $|b-a|\ge\varepsilon$ ならば (1) に戻る。そうでなければ c を解として出力する。

注意 E.1 (反復の停止のための ε) 目標とする誤差としては、C 言語で倍精度浮動小数点数 `double` (実習で利用している C コンパイラーでは、相対精度が 10 進数に換算して 16 桁弱) を用いる場合は[18]解の絶対値の推定値 (本当に大雑把な、桁数の目安がつく位のもので構わな

[18]一方、単精度 (float) の場合には 10^{-7} 程度にすべきであろう。

い) の大きさに 10^{-15} をかけた数程度にするのが適当であろう[19]。それ以上小さく取っても、使用している浮動小数点数の体系の能力を越えてしまうことになり、意味がない。この問題の場合は解は区間 $(0,1)$ の真ん中位にあるので、ざっと 1 程度ということで、$\varepsilon = 10^{-15}$ 位が良いだろう (この数を表すのに、C 言語では "`1.0e-15`" と書く)。

E.2 Newton 法

非線形方程式を解くためのもう一つの代表的な方法が **Newton** 法である。
これは f が微分可能な関数で、方程式 $f(x) = 0$ の近似解 x_0 が得られている時、漸化式

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (n = 0, 1, 2, \cdots)$$

で数列 $\{x_n\}_{n=0,1,2,\cdots}$ を定めると、適当な条件[20]の下で

$$\lim_{n \to +\infty} x_n = x_*$$

と収束し、極限 x_* は方程式の解になっている:

$$f(x_*) = 0$$

ということを利用したもので、実際のアルゴリズムは次のようになる。

Newton 法のアルゴリズム

(1) 適当な初期値 x_0 を選ぶ。

(2) $x \leftarrow x_0$

(3) $x \leftarrow x - f(x)/f'(x)$ とする。

(4) まだ近似の程度が十分でないと判断されたら (3) に戻る。そうでなければ x を解として出力する。

E.3 二分法 vs. Newton 法

ここで紹介した二つの方法はどちらが優れているだろうか？それぞれの長所・短所をあげて比較してみよう。

E.3.1 二分法の特徴

1. f が微分可能でなくとも連続でありさえすれば適用できる。

[19] 本当は、もっと精密な考えに基づいた妥当なやり方があるが、ここでは省略する。興味があれば、杉原正顯、室田一雄、『数値計算法の数理』、岩波書店などを見よ。

[20] Newton 法が収束するための十分条件は色々知られているが、ここでは説明しない。簡単なものは微分積分学のテキストに載っていることも多い。

2. f は 1 変数実数値関数でない場合は適用が難しい (特に実数値であることはほとんど必要であると言ってよい)。

3. $f(\alpha)f(\beta) < 0$ なる α, β が見つかっていれば、確実に解が求まる。

4. 収束はあまり速くない。1 回の反復で 2 進法にして 1 桁ずつ精度が改善されていく程度である。

E.3.2 Newton 法の特徴

1. 適用するには少なくとも f が微分可能である必要がある。

2. 微分可能であっても f の実際の計算が難しい場合は適用困難になる。

3. f は多変数ベクトル値関数でも構わない (それどころか無限次元の方程式にも使うことが出来る)。

4. 適切な初期値を探すことは、場合によってはかなり難しい。

5. 求める解が重解でない場合には、十分真の解に近い初期値から出発すれば 2 次の収束となり (合っている桁数が反復が一段進むごとに 2 倍になる)、非常に速い。

E.3.3 とりあえずの結論

総合的に見て「まずは Newton 法を使うことを考えよ、それが困難ならば二分法も考えてみよ。」というところだろうか。

F 例題を解くプログラム例

二分法のプログラム bisection.c[21], newton.c[22] は情報処理 II の WWW ページ `http://www.math.meiji.ac.jp/~mk/syori2/` から入手できる。WWW ブラウザー「ファイル・メニュー」の「名前をつけて保存」を用いてセーブする。2004 年度情報科学センターの教育用情報処理室の Windows 環境からは、z: ドライブ (UNIX 環境のホームディレクトリィ) にセーブするのが便利であろう。

F.1 Newton 法の場合

1 を初期値として、許容精度 10^{-15} を指示して Newton 法で解かせたのが以下の結果 (入力は `1 1e-15`)。

[21] `http://www.math.meiji.ac.jp/~mk/syori2/cprog/bisection.c`
[22] `http://www.math.meiji.ac.jp/~mk/syori2/cprog/newton.c`

```
samba01% gcc -o newton newton.c -lm
samba01% ./newton
 初期値 x0, 許容精度ε =1 1e-15
f(   0.750363867840244)=-1.89e-02
f(   0.739112890911362)=-4.65e-05
f(   0.739085133385284)=-2.85e-10
f(   0.739085133215161)= 0.00e+00
f(   0.739085133215161)= 0.00e+00
samba01%
```

注意 **F.1** Newton 法の繰り返しを停止させるための良いルールを独力で発見するのはかなり難しい。上の例題プログラムの採用したルールは、多くのプログラムで採用されているやり方ではあるが、いつでもうまく行く方法とは言えない。現時点で「これが良い方法」と見なされている方法は一応あるが、結構複雑なのでここでは説明しない。

F.2 二分法の場合

区間 $(0, 1)$ 内に解があることがわかるから、二分法で許容精度 10^{-15} を指示して解かせたのが以下の結果 (入力は 0 1 1e-15)。関数値が、区間の左端では正、右端では負になったまま区間が縮小して行くのを理解しよう。

```
samba01% gcc -o bisection bisection.c -lm
samba01% ./bisection
 探す区間の左端α , 右端β , 許容精度ε =0 1 1e-15
f(   0.000000000000000)= 1.00e+00, f(   1.000000000000000)=-4.60e-01
f(   0.500000000000000)= 3.78e-01, f(   1.000000000000000)=-4.60e-01
f(   0.500000000000000)= 3.78e-01, f(   0.750000000000000)=-1.83e-02
f(   0.625000000000000)= 1.86e-01, f(   0.750000000000000)=-1.83e-02
f(   0.687500000000000)= 8.53e-02, f(   0.750000000000000)=-1.83e-02
f(   0.718750000000000)= 3.39e-02, f(   0.750000000000000)=-1.83e-02
f(   0.734375000000000)= 7.87e-03, f(   0.750000000000000)=-1.83e-02
f(   0.734375000000000)= 7.87e-03, f(   0.742187500000000)=-5.20e-03
f(   0.738281250000000)= 1.35e-03, f(   0.742187500000000)=-5.20e-03
f(   0.738281250000000)= 1.35e-03, f(   0.740234375000000)=-1.92e-03
f(   0.738281250000000)= 1.35e-03, f(   0.739257812500000)=-2.89e-04
f(   0.738769531250000)= 5.28e-04, f(   0.739257812500000)=-2.89e-04
f(   0.739013671875000)= 1.20e-04, f(   0.739257812500000)=-2.89e-04
f(   0.739013671875000)= 1.20e-04, f(   0.739135742187500)=-8.47e-05
f(   0.739074707031250)= 1.74e-05, f(   0.739135742187500)=-8.47e-05
f(   0.739074707031250)= 1.74e-05, f(   0.739105224609375)=-3.36e-05
f(   0.739074707031250)= 1.74e-05, f(   0.739089965820312)=-8.09e-06
f(   0.739082336425781)= 4.68e-06, f(   0.739089965820312)=-8.09e-06
f(   0.739082336425781)= 4.68e-06, f(   0.739086151123047)=-1.70e-06
f(   0.739084243774414)= 1.49e-06, f(   0.739086151123047)=-1.70e-06
f(   0.739084243774414)= 1.49e-06, f(   0.739085197448730)=-1.08e-07
f(   0.739084720611572)= 6.91e-07, f(   0.739085197448730)=-1.08e-07
f(   0.739084959030151)= 2.92e-07, f(   0.739085197448730)=-1.08e-07
f(   0.739085078239441)= 9.20e-08, f(   0.739085197448730)=-1.08e-07
f(   0.739085078239441)= 9.20e-08, f(   0.739085137844086)=-7.75e-09
f(   0.739085108041763)= 4.21e-08, f(   0.739085137844086)=-7.75e-09
f(   0.739085122942924)= 1.72e-08, f(   0.739085137844086)=-7.75e-09
f(   0.739085130393505)= 4.72e-09, f(   0.739085137844086)=-7.75e-09
f(   0.739085130393505)= 4.72e-09, f(   0.739085134118795)=-1.51e-09
f(   0.739085132256150)= 1.61e-09, f(   0.739085134118795)=-1.51e-09
f(   0.739085133187473)= 4.63e-11, f(   0.739085134118795)=-1.51e-09
f(   0.739085133187473)= 4.63e-11, f(   0.739085133653134)=-7.33e-10
f(   0.739085133187473)= 4.63e-11, f(   0.739085133420303)=-3.43e-10
f(   0.739085133187473)= 4.63e-11, f(   0.739085133303888)=-1.48e-10
f(   0.739085133187473)= 4.63e-11, f(   0.739085133245680)=-5.11e-11
f(   0.739085133187473)= 4.63e-11, f(   0.739085133216577)=-2.37e-12
f(   0.739085133202025)= 2.20e-11, f(   0.739085133216577)=-2.37e-12
f(   0.739085133209301)= 9.81e-12, f(   0.739085133216577)=-2.37e-12
f(   0.739085133212939)= 3.72e-12, f(   0.739085133216577)=-2.37e-12
f(   0.739085133214758)= 6.74e-13, f(   0.739085133216577)=-2.37e-12
f(   0.739085133214758)= 6.74e-13, f(   0.739085133215667)=-8.48e-13
f(   0.739085133214758)= 6.74e-13, f(   0.739085133215212)=-8.66e-14
f(   0.739085133214985)= 2.94e-13, f(   0.739085133215212)=-8.66e-14
f(   0.739085133215099)= 1.04e-13, f(   0.739085133215212)=-8.66e-14
f(   0.739085133215156)= 8.55e-15, f(   0.739085133215212)=-8.66e-14
f(   0.739085133215156)= 8.55e-15, f(   0.739085133215184)=-3.91e-14
f(   0.739085133215156)= 8.55e-15, f(   0.739085133215170)=-1.53e-14
f(   0.739085133215156)= 8.55e-15, f(   0.739085133215163)=-3.44e-15
f(   0.739085133215159)= 2.55e-15, f(   0.739085133215163)=-3.44e-15
f(   0.739085133215159)= 2.55e-15, f(   0.739085133215161)=-4.44e-16
f(   0.739085133215160)= 1.11e-15, f(   0.739085133215161)=-4.44e-16
f(   0.739085133215160)=1.110223e-15
samba01%
```

G　レポート課題 6

(これは昨年度の講義資料を参考のために見せているだけであって、ここに書いてあるのが今年度の課題ではないことに注意せよ。)

以下から二つ以上 (好きなだけ) 解いて、提出せよ。〆切は 7 月 14 日 (?)。

課題 6-1

色々な方程式を二分法、Newton 法で解いてみよ。初期値の選び方を変えて、収束するかしないか、試しなさい。収束の判定条件には注意を払うこと (特に理由なく低い精度の答を出しただけの場合は減点)。最終的に得られる精度や、必要な反復の回数はどの程度になるか。Newton 法の収束のための十分条件を本などで探すことができたら、それも説明すること。

課題 6-2

平方根 $\sqrt{a}$ や立法根 $a^{1/3}$ を方程式の解の形に定式化して、Newton 法で解いて見よ。その結果を C 言語のライブラリィ関数 `sqrt()` や `pow()` [23] で計算した値と比較してみよ。

ここでは解説しないが、初等関数などもコンピューターの中では、四則演算などの簡単な演算の組合せで計算されていることが多い。

課題 6-3

C 言語のライブラリィ関数 `asin()`, `acos()`, `atan()` は使わずに、arcsin, arccos, arctan を計算するプログラムを作り、実験せよ。

要は、逆関数の計算にも使える (y が与えられているとして、方程式 $f(x) = y$ を x について解けば、$x = f^{-1}(y)$ が得られるはず) ということである。

課題 6-4

連立方程式[24]

$$\begin{aligned} x^2 - y^2 + x + 1 &= 0 \\ 2xy + y &= 0 \end{aligned}$$

を Newton 法を用いて解くプログラムを作れ。ヒント:

$$\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix},$$

$$f(\vec{x}) = \begin{pmatrix} x^2 - y^2 + x + 1 \\ 2xy + y \end{pmatrix}$$

[23] 参考情報: `pow(a,b)` で `a` の `b` 乗が計算できる。例えば `pow(2.0, 1.0/3.0)` で 2 の立法根が計算できる。

[24] この問題は複素数の世界で考えると、ネタがばれる。

とおくと、方程式は $f(\vec{x}) = 0$ と書ける。f の $\vec{x}$ における Jacobi 行列を $f'(\vec{x})$ とすると、Newton 法の式は

$$\vec{x}_{n+1} = \vec{x}_n - [f'(\vec{x}_n)]^{-1} f(\vec{x}_n)$$

となる。初期値 $\vec{x}_0$ を $\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ として実験せよ。

A Newton 法の意味

Newton 法の式の意味を簡単に説明しよう。微分の定義によると、x が a に十分近いところでは、f は「接線の式」で近似されることが期待される:

$$f(x) \fallingdotseq f'(a)(x-a) + f(a).$$

今 a が $f(x) = 0$ の解に十分近いとすると、$f(x) = 0$ の代わりに

$$f'(a)(x-a) + f(a) = 0$$

を解くことにより、a よりも精度の高い近似解が得られると考えるのは自然であろう。実際に実行すると、まず移項して

$$f'(a)(x-a) = -f(a).$$

両辺に $[f'(a)]^{-1}$ をかけて

$$x - a = -[f'(a)]^{-1} f(a),$$

ゆえに

$$x = a - [f'(a)]^{-1} f(a) = a - \frac{f(a).}{f'(a)}.$$

多変数の場合も、$[f'(a)]^{-1}$ を Jacobi 行列の逆行列と考えれば、まったく同様に Newton 法が使える。