第8章　C言語 基礎編：定番プログラム

8.1　ニュートン法

この章では、プログラミングを学習すると必ずと言ってよいほど目に触れる有名なアルゴリズムについて紹介します[1] (具体的なプログラムの作成は読者に任せます)。

1 変数の関数 $f(x)$ が与えられ、$f(x)$ が微分可能であるとします。このとき、$f(x) = 0$ の近似解 (数値解) を求める方法として有名なのが**ニュートン法**[2] (Newton's method) です。ニュートン法は非線形方程式を解く非常に有効な手段です。ニュートン法の計算式は、数列

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad (k = 0, 1, 2, \cdots)$$

で与えられます。

ニュートン法の原理を解説しながら、計算式を導きましょう。まず、図 8.1 のように適切な初期値 x_0 を選びます。次に、点 $(x_0, f(x_0))$ で $y = f(x)$ の接線

$$y = f'(x_0)(x - x_0) + f(x_0)$$

を引きます。更に、この接線と x 軸の交点の x 座標

$$x\ (= x_1) = x_0 - \frac{f(x_0)}{f'(x_0)}$$

---

[1] 5.1 節で紹介した「平方根を求めるアルゴリズム」や「ユークリッドの互除法を用いて最大公約数を求めるアルゴリズム」も非常に有名です。

[2] ニュートン・ラフソン (Raphson) 法とも呼ばれます。また、セカント法と呼ばれ、$f'(x_k)$ の代わりに差分 $\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$ で近似する方法もあります。

を求め、これを x_1 と置きます。同様に、上記の手順を繰り返し行なうことでニュートン法の計算式

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad (k = 0, 1, 2, \cdots)$$

を得ます。また、図 8.1 より、計算式から得られる数列の値が $f(x) = 0$ の解 α に近づく様子が見て取れます。

注意：数列の値が振動し収束しない場合があるので、解の値にめぼしを付け初期値には解に近い値を選ぶようにする (収束しない場合もあるので反復回数に制限を設けておく)。

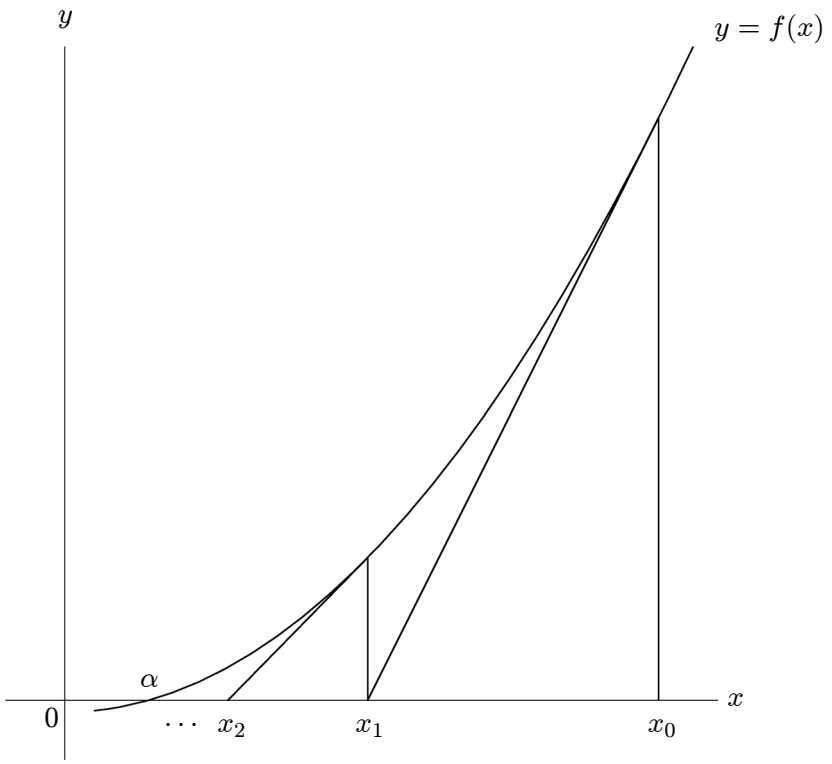

図 8.1: ニュートン法による解の近似

問題 1　ニュートン法を用いて 2 の平方根 $\sqrt{2}$ $(= \alpha)$ の近似値を求めるプログラムを作成しなさい (ヒント：$\alpha = 1.41\cdots$)。

問題 2　ニュートン法を用いて 2 の立方根 $\sqrt[3]{2}$ $(= \alpha)$ の近似値を求めるプログラムを作成しなさい (ヒント：$\alpha = 1.25\cdots$)。

問題 3　ニュートン法を用いて $x - \cos x = 0$ の解 α の近似値を求めるプログラムを作成しなさい (ヒント：$\alpha = 0.73\cdots$)。

8.2 モンテカルロ法

乱数を用いて数学や物理の問題を解くことを**モンテカルロ法**[3] (Monte Carlo methods) といいます[4]。例として、モンテカルロ法を用いて円周率 π の値を求めてみましょう。図 8.2 のように、正方形領域 $S = \{(x,\ y) \mid 0 \le x \le 1,\ 0 \le y \le 1\}$ と円盤領域 $C = \{(x,\ y) \mid x^2+y^2 \le 1,\ x \ge 0,\ y \ge 0\}$ とします。このとき、乱数を使って平方領域 S に一様に点をばらまくと、

$$(\text{正方形領域 } S \text{ に含まれる点の数}) : (\text{円盤領域 } C \text{ に含まれる点の数}) \ = \ 1 : \frac{\pi}{4}$$

が成り立ち、π の近似値を求めることができます。

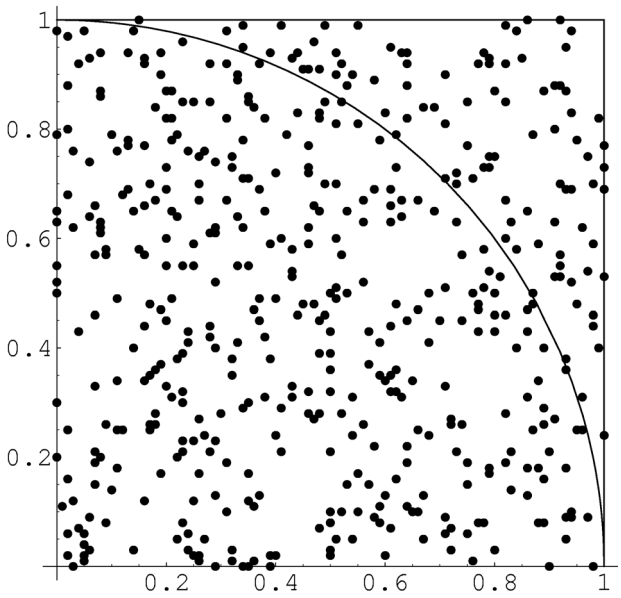

図 8.2: モンテカルロ法による円周率 π の近似値

問題 1 モンテカルロ法を用いて円周率 π の近似値を求めるプログラムを作成しなさい。

[3]モンテカルロ法は、1777 年に発表されたビュフォン (Buffon) の針の問題に端を発し、モンテカルロ法という名前は、1949 年にメトロポリス (Metropolis) とウラム (Uram) がギャンブル場で有名なモナコの町名にちなんで命名されました。

[4]モンテカルロ法は、主に数値積分に用いられ、特に、多次元の数値積分などに有効です。また、モンテカルロ法を用いると積分の形に表せない複雑な問題なども比較的簡単に解ける場合があります。

8.3　エラトステネスのふるい

エラトステネスのふるい (sieve of Eratosthenes) は、素数を列挙する高速なアルゴリズムとしてよく知られています。例として、エラトステネスのふるいを使って、1 から 30 までに含まれる素数列を求めてみましょう[5]。まず、1 から 30 までの数を列挙します (1 は素数ではないので、最初から消しておく)。

×1,　2,　3,　4,　5,　6,　7,　8,　9,　10,
11,　12,　13,　14,　15,　16,　17,　18,　19,　20,
21,　22,　23,　24,　25,　26,　27,　28,　29,　30

次に、消されていない数で 1 の次に大きな数 2 の倍数を全て消します。

×1,　2,　3,　×4,　5,　×6,　7,　×8,　9,　×10,
11,　×12,　13,　×14,　15,　×16,　17,　×18,　19,　×20,
21,　×22,　23,　×24,　25,　×26,　27,　×28,　29,　×30

更に、消されていない数で 2 の次に大きな数 3 の倍数を全て消します。

×1,　2,　3,　×4,　5,　×6,　7,　×8,　×9,　×10,
11,　×12,　13,　×14,　×15,　×16,　17,　×18,　19,　×20,
×21,　×22,　23,　×24,　25,　×26,　×27,　×28,　29,　×30

同様に、この操作を繰り返すことで、1 から 30 までに含まれる素数列を求めることができます (×の付いていない数が素数)。

×1,　2,　3,　×4,　5,　×6,　7,　×8,　×9,　×10,
11,　×12,　13,　×14,　×15,　×16,　17,　×18,　19,　×20,
×21,　×22,　23,　×24,　×25,　×26,　×27,　×28,　29,　×30

⇓

$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}$

なお、30 以下の素数列を求めるのであれば、$[\sqrt{30}] = [5.47\cdots] = 5$ 以下の素数の倍数を消せば十分です (例えば、$30 = 5 \times 6$ であるから、$5 + 1$ で割る以前に 5 で割れてしまうため)。

* $[x]$ は x を超えない最大の整数を表す (ガウス記号)。

問題 1　エラトステネスのふるいを使って N までの素数列を求めるプログラムを作成しなさい。

問題 2　問題 1 のプログラムをポインタを使ったプログラムに直しなさい。

問題 3　N までの全ての数を列挙すると 2 の倍数が配列の半分を占めメモリの無駄が多い。最初から 2 の倍数を除き、エラトステネスのふるいを使って N までの素数列を求めるプログラムを作成しなさい。

[5] 全ての数を列挙すると、2 の倍数がその半分を占めるためメモリの無駄が多くなります。これを改善するため、最初から 2 の倍数を除いて $2n + 1$ $(n = 1, 2, \cdots)$ を対象にエラトステネスのふるいを実行するようにするとよい。更に効率よくするには、2 の倍数と 3 の倍数を除いて $6n \pm 1$ $(n = 1, 2, \cdots)$ を対象にするとよい。