電気303/電情303 数値解析 (2)

非線形方程式と非線形最小化問題の数値解法

非線形方程式

- 工学的な問題を数学的に解くためには, 解くべき問題を数式によって表現する必要がある.
- 問題の表現にはさまざまな形があるが, 変数 (1 個でも複数でもよい) の関係が等号であらわされているものを方程式, 不等号であらわされているものを不等式という.

- 式の中に変数の微分が含まれている場合には「微分」, 式が線形であるばあいには「線形」, 非線形である場合には「非線形」という修飾語が付く.

- たとえば…

線形方程式	vs	非線形方程式
線形微分方程式	vs	非線形微分方程式
線形不等式	vs	非線形不等式

- 例として, 「体積が 1 となる球の半径を求めよ」という問題を考える.
- この問題は, 非線形方程式で表現される.

- 半径が r の球の体積の公式は

$$\frac{4\pi r^3}{3}$$

であるから,

$$r = \sqrt[3]{\frac{3}{4\pi}}$$

が求める解である.

- 上記のように解が解析的に表現できる問題は稀で, 大抵は数値的な近似解しか求められない.
- 以下の議論では, 数値的な近似解を求める手法を考えてゆく.

- 1 個の実変数 x に関する単一の方程式は,

$$f(x) = 0$$

という形で書ける.

- 解きたい方程式が

$$g(x) = c$$

(ただし c は定数) である場合, これを

$$g(x) - c = 0$$

と書き直す.

- 更に,

$$f(x) = g(x) - c$$

とおけば, 先に述べた形になる.

- 非線形方程式を解くことは, 関数のグラフと x 軸の交点を求めることに対応する.
- 交点がただひとつの場合は比較的単純.
- 交点がたくさんある場合は初期値に関する注意が必要. 解をすべて求めるのは易しくない.

- 1 変数であれば, 関数のグラフを描画すれば, x 軸との交点を求めることで非線形方程式の近似解が得られるのであるが…

- そもそも論として, グラフの描画自体が数値計算の結果なので, これは問題の解決になっていないし…
- 関数値の評価自体の計算量が多い関数では, 描画に極めて手間がかかる.
- 更に, グラフから読み取った交点の精度は高くない.

- 2 変数関数ではこの方法は極めて使いにくいし, 3 変数関数以上ではそもそもグラフが描けない.

- そこで, 視覚に頼るような不確かな方法ではなく, もっとコンピュータに適した形で求解をおこないたい.

- 求解では, より少ない関数値の評価回数で高精度の近似解が求められることが望ましい.

- 以下の議論では, 解くべき方程式の解を真の解と呼び, 数値計算によって求められた近似解と区別する.

- 非線形方程式を解くための解法はふつうは反復解法である.
- 反復解法とは, 適切な初期値から出発し, 何度も近似解を改善することによって, 近似解を真の解に収束させる解法である.

- 線形問題に対する解法には, 万能に近いものが多い.

- これに対し, 非線形問題に対する解法には得手不得手がある
  - ▷ 応用範囲が広い解法は効率が悪く, 応用範囲が狭い解法は効率が良いという傾向がある.
  - ▷ 問題に適した解法を選ぶ必要がある..

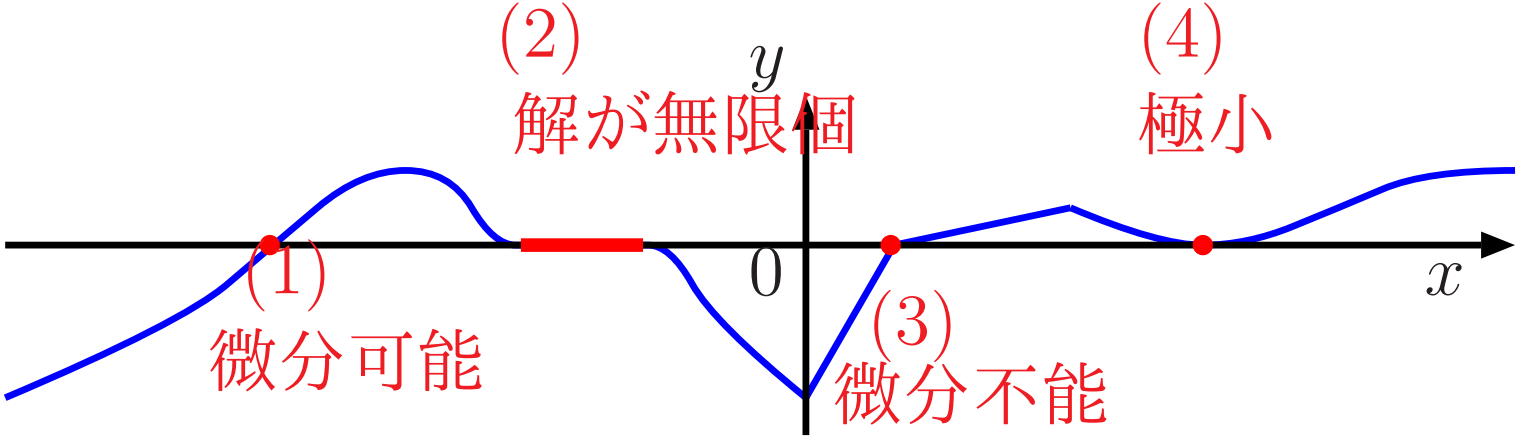
- 解くべき非線形方程式を

$$f(x) = 0$$

とする. f は実変数の実数値関数である.

- 非線形方程式の解はたくさんあることもある.
- 以下では, ある解 x_* の近傍における関数の振舞いについて考える.

- 解には, 以下の図のようなパターンがある.



- 先の図の (1) から (3) までの場合を一応カバーできるのが 2 分法と呼ばれる解法で, 汎用性が高く, $f(x)$ が単調関数ならつねに収束するが, 収束が遅いという欠点がある.

- 先の図の (4) は最小化問題として解く必要がある.

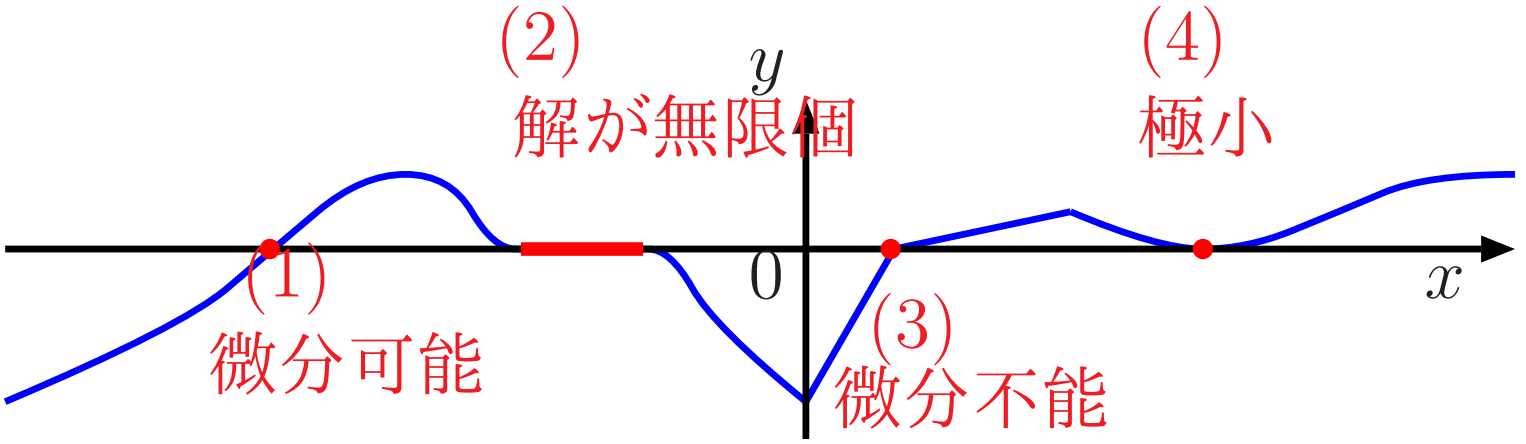
- 高速で実用的なのは Newton 法であるが, 先の図の (1) の場合にしか対応できない.

- 素朴な形の Newton 法は高速であるが, 初期値が真の解の近くにないと発散する可能性がある.

- 今日では, とくに多変数の場合には, 可変ステップ幅の Newton 法と呼ばれる方法が使われることが普通. ステップ幅の決定には, 直線探索あるいは信頼領域法と呼ばれる手法が使われる. こちらは, 一定の条件のもとで, 初期値によらず真の解に収束する (この講義では深入りしない).

二分法

- まず先の図を再掲する.



- (1) から (3) までに共通するのは, $f(x)$ が真の解の近傍で単調関数であるということ.
- まず, $f(x)$ は単調増加関数である場合について述べる.
- 単調減少関数の場合も考え方は同様であるが, こちらについては後で改めて述べる.

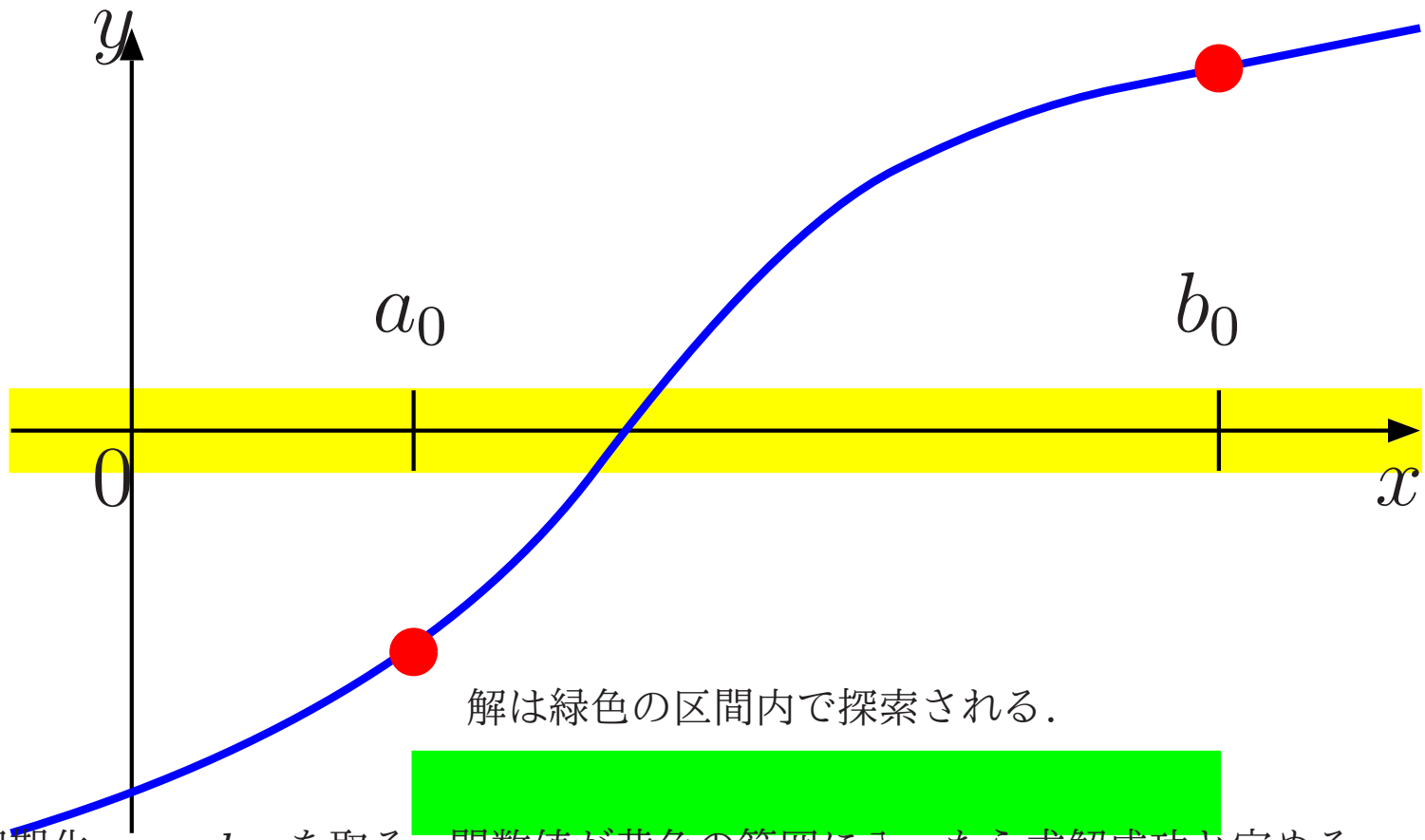
- 二分法のアルゴリズムは以下に述べるようなものである.

- **初期化:** $f(a_0) < 0,\ f(b_0) > 0$ を見たす a_0, b_0 を何らかの方法で見付ける. $k = 0$ とする. 誤差の許容値 $\varepsilon > 0$ を定める.

- **ループ:** $c_k = \dfrac{a_k + b_k}{2}$ とし, $f(c_k)$ を評価し, 続いて以下を順に実行する.

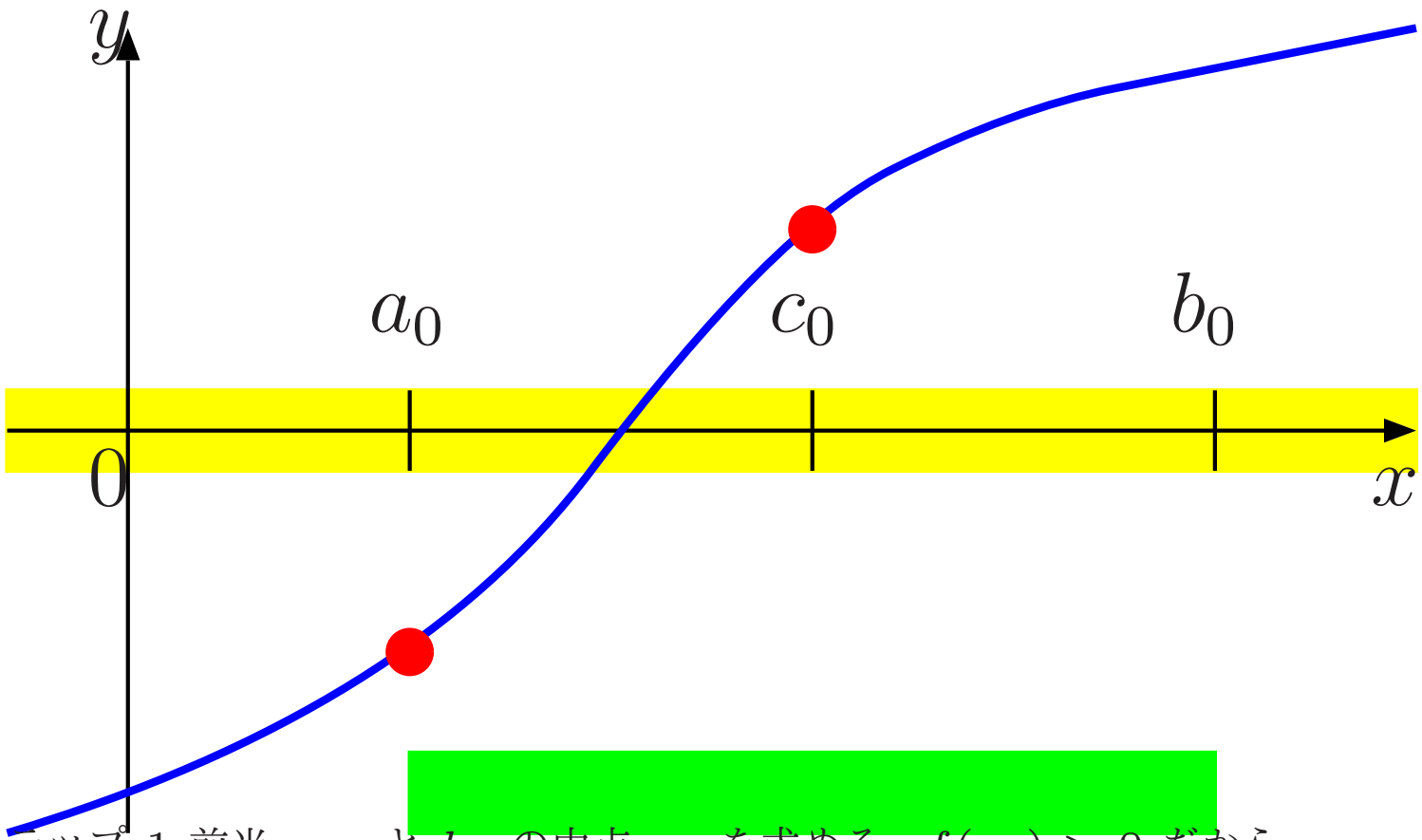
- $|f(c_k)| < \varepsilon$ なら c_k が近似解 (終了).
- $|f(c_k) \geq \varepsilon$ のときは:
  - $f(c_k) > 0$ なら解は区間 $[a_k, c_k]$ にあるので, $a_{k+1} = a_k, b_{k+1} = c_k$ とする.
  - $f(c_k) < 0$ なら解は区間 $[c_k, b_k]$ にあるので, $a_{k+1} = c_k, b_{k+1} = b_k$ とする.
  - k に 1 を加えてループ先頭に戻る

- 図で計算の過程を見てゆく.

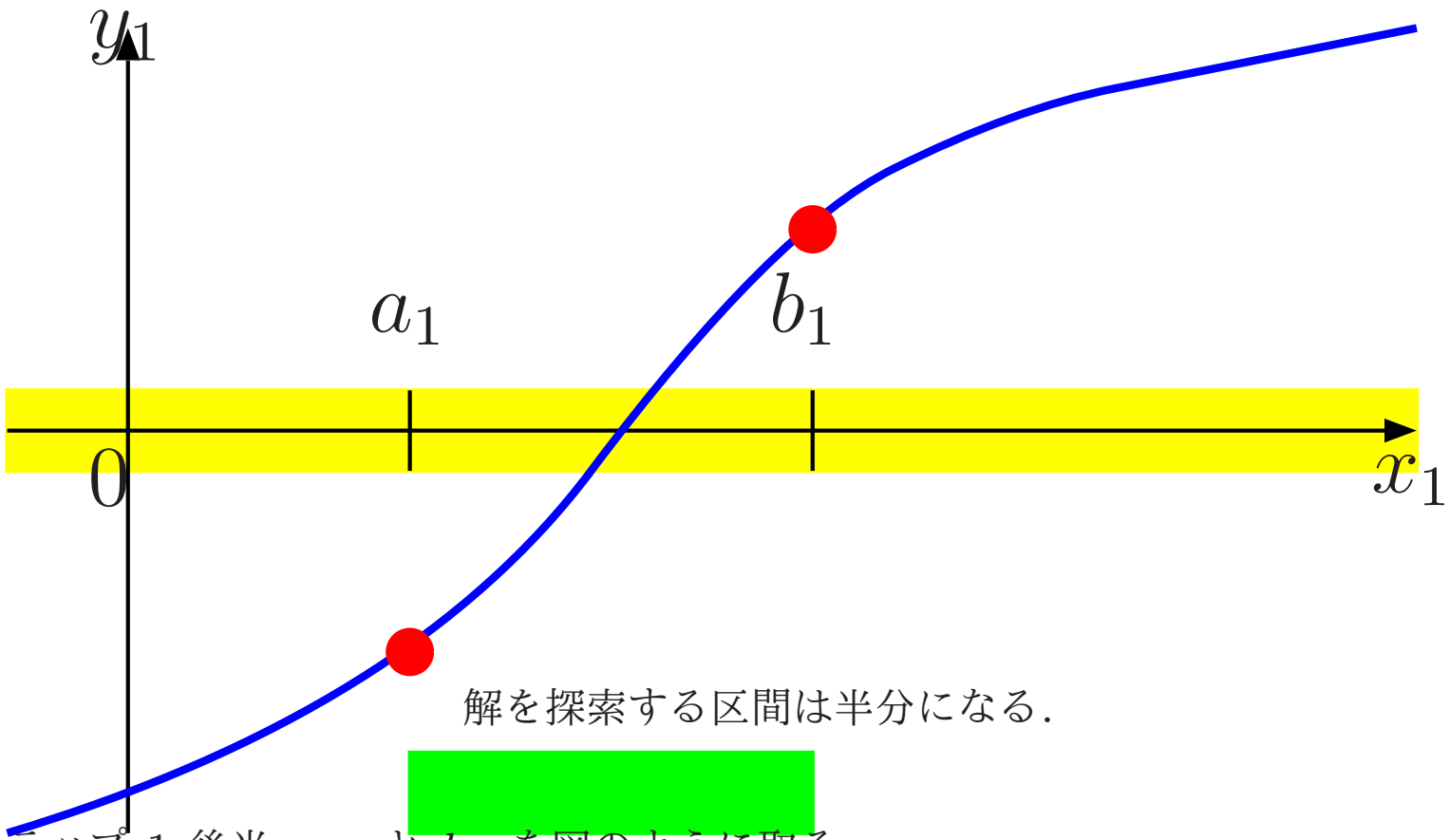


解は緑色の区間内で探索される．

初期化．a_0，b_0 を取る．関数値が黄色の範囲に入ったら求解成功と定める．

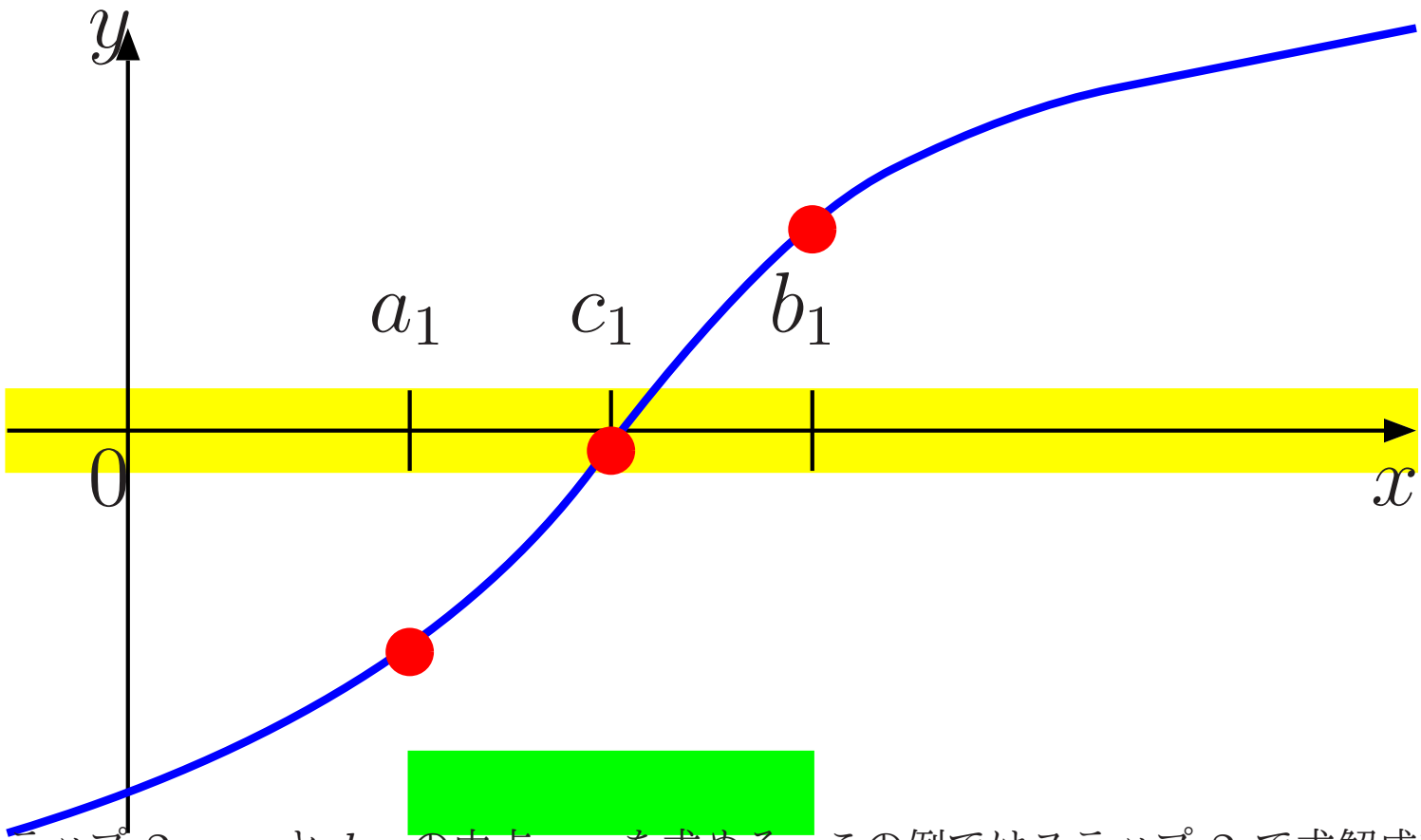


ステップ 1 前半. a_0 と b_0 の中点 c_0 を求める. $f(c_0) > 0$ だから $\cdots$



解を探索する区間は半分になる.

ステップ 1 後半. a_1 と b_1 を図のように取る.



ステップ 2. a_1 と b_1 の中点 c_1 を求める. この例ではステップ 2 で求解成功.

- 一般には, もっとずっと多くの繰り返しが必要だが, 考え方は同じ.

- 次に $f(x)$ が単調減少の場合を考える.

- 初期化:
  - $f(a_0) > 0$, $f(b_0) < 0$ を満たす a_0, b_0 を何らかの方法で見付ける.
  - $k = 0$ とする.
  - 誤差の許容値 $\varepsilon > 0$ を定める.

- **ループ:** ステップ 1, ステップ 2, . . . , ステップ k, . . . というふうに, 以下の計算を繰り返す.

- $[a_k, b_k]$ の中点 $c_k = \dfrac{a_k + b_k}{2}$ を求める.
- $|f(c_k)| < \varepsilon$ なら終了.
- $f(c_k) \leq -\varepsilon$ なら $[a_{k+1}, b_{k+1}] = [a_k, c_k]$ としステップ $k+1$ に進む.
- $f(c_{k+1}) \geq \varepsilon$ なら $[a_{k+1}, b_{k+1}] = [c_k, b_k]$ としステップ $k+1$ に進む.

二分法のバリエーションと拡張

- c_{k+1} を $[a_k, b_k]$ の中点とするかわりに, 点 $(a_k, f(a_k))$ と点 $(b_k, f(b_k))$ を結ぶ線分が x 軸と交わる点とする方法がある (はさみうち法)
- はさみうち法は, c_{k+1} を求める方法が 2 分法と違うだけで, 他の部分は同じ.

- 実は, はさみうち法はNewton法の近似であるセカント法とよく似ている.
- プログラムは教科書にある. 著者のホームページからダウンロードできるので, 試してみるとよい.

- 非線形方程式

$$e^x - 10 = 0$$

を 2 分法およびはさみうち法によって解くプログラム例とその実行結果を次ページに示す. ただし, 初期値を, $a = 0, b = 3$ とし, 終了条件を,

$$|e^x - 10| < 10^{-3}$$

としている.

2 分法のプログラム例

```
deff('y=f(x)','y=exp(x)-10');
err=1.0E-3;a=0;b=3;k=0;
while(1)
  c=(a+b)/2;
  v=f(c);
  if(abs(v)<err)
    break;
  else
    if(v>err)
      b=c;
    else
      a=c;
    end
  end
  k=k+1;
end
```

2 分法の $f(x)$ の履歴

繰り返し回数	$f(x)$
1	-5.518311
2	-0.512264
3	3.804574
4	1.444394
5	0.420239
6	-0.056938
7	0.178855
8	0.060267
9	0.001493
10	-0.027765
11	-0.013147
12	-0.005830
13	-0.002169

はさみうち法のプログラム例

```
deff('y=f(x)','y=exp(x)-10');
err=1.0E-3;a=0;b=3;k=0;
while(1)
  c=a+(b-a)*abs(f(a))/(f(b)-f(a));
  v=f(c);
  if(abs(v)<err)
    break;
  else
    if(v>err)
      b=c;
    else
      a=c;
    end
  end
  k=k+1;
end
```

はさみうち法の $f(x)$ の履歴

繰り返し回数	$f(x)$
1	-5.884815
2	-2.619458
3	-0.927388
4	-0.299431
5	-0.093712
6	-0.029040
7	-0.008971
8	-0.002769

- この例では, 終了条件 ($|f(x)| < 10^{-3}$) を満たすために, 2 分法では 13 回の繰り返しがなされているのに対し, はさみうち法では 8 回の繰り返しがなされている.

- 教科書では $f(x, y) = 0$, $g(x, y) = 0$ という 2 変数非線形連立方程式を二分法で解く手法が解説されているが…
- この方法は, $f(x, y)$ と $g(x, y)$ のどちらか一方がある変数について解けることを前提にしている

- たとえば $f(x, y) = 0$ が $y = h(x)$ と書き直せるなら, 解くべき非線形方程式は $g(x, h(x)) = 0$ となり, 1 変数の二分法が適用できる.
- 数学的には, $f(x, y) = 0$ を局所的に $y = h(x)$ と書き直せるための条件はわかっている (陰関数定理を使う).

- 数値解析の観点から言うと, $f(x, y) = 0$ という式から $y = h(x)$ という式を解析的に求めるのは, 余程簡単な問題でなければ無理.
- 以上の理由により, 2 変数の 2 分法は実用性が乏しいと思われるので, この講義では説明を省略する.

計算結果の表記について

- 結果を表記するときには精度に注意
- 倍精度の浮動小数点数では, 仮数部の最小桁は $2^{-52} \simeq 2.2 \times 10^{-16}$ なので, 仮数部の有効数字は 15 桁程度.

- 計算の過程でさらに精度が落ちていることもあり得る.
- コンピュータが表示した数字の桁を全部記録することには必ずしも意味はない.

Newton 法

- Newton 法には, 非線形方程式を 解くためのものと非線形最小化問題を解くためのものがある.
- 教科書で取り扱われているのは非線形方程式を解くための Newton 法のみなので, 当面は, 非線形方程式を解くための Newton 法について説明する.

- Newton 法は, 非線形方程式を線形近似し, 近似した線形方程式を解く手順を繰り返す方法.
- 単純な Newton 法は, 初期値が真の解に十分近くないと発散することがある.
- 今日では, 発散を防ぐ方法が色々と知られている.

- 微分可能な 1 変数実数値関数関数 f に関する方程式 $f(x) = 0$ を解きたい.
- Newton 法が適用できるためには, f の導関数が零にならないことが必要.
- f の導関数が零になる場合には, 他の解法 (2 分法など) を使う必要がある.

- 初期値 x_0 が与えられているものとする.
- $f(x_0) = 0$ なら x_0 が解であり, これ以上計算する必要はない. そこで, $f(x_0) \neq 0$ の場合についてのみ考える.

- 数値解を x_0 から $x_0 + d$ に変更したとき, テイラー展開によって関数値の変化を近似すると,

$$f(x_0 + d) \simeq f(x_0) + f'(x_0)d$$

となる. ただし, f' は f の導関数である.

- 線形近似の精度が良ければ,

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

とすることにより, 関数値は零に近付く筈である. この考えに基づいて繰り返し計算をおこなうのがNewton 法である.

- アルゴリズムの形で書き下すと次ページのようになる.

- **初期化:** 初期値 x_0 と誤差の許容値 $\varepsilon > 0$ を定める. $k = 0$ とする.
- **ループ:** $|f(x_k)| < \varepsilon$ であれば終了. そうでない場合には,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

とし, k に 1 を加えてループの先頭に戻る.

- 教科書のサンプルプログラムには, x_{k+1} と x_k の差が大きすぎるときには発散と見做して初期値を選び直す部分と, 繰り返し回数が一定を超えたら発散と見做して初期値を選び直す部分が含まれる.
- 単純な Newton 法を使う場合には, 上記のような工夫が必要.

- 単純な Newton 法には, 解が高速に得られる (2 次収束) という長所がある一方で, 初期値の取り方しだいで発散することがあるという問題がある.
- 2 分法と異なり, Newton 法は素直に多変数の問題に拡張できる.

- Newton 法の収束に関する数学の定理は繁雑なので, この講義では取り扱わない.

- 今日では, 1 変数, 多変数の場合の双方について, 関数 f が一定の条件を満たすとき, 初期値によらず解に収束する Newton 法が知られている (1990 年前後に確立).

- この手法は直線探索法と信頼領域法に大別される. いずれも複雑なのでこの講義では名前を紹介するだけだが, 数値計算ソフトには実装されていることが多い.
- 大域的に収束する Newton 法を使えば, 初期値を選び直す作業は不要.

セカント法

- Newton 法において, $f'(x_k)$ を

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

で近似したものを, セカント法と呼ぶ.

- 微分の定義に戻って考えれば, 上記の式が $f'(x_k)$ の近似になっていることがわかる.
- 式をよく見るとわかるが, セカント法は, はさみうち法と似ている.

- セカント法は, 関数の微分の評価が難しい (計算量などの観点から) ときに用いられる. ただし, その収束の速さは Newton 法に劣ることがふつうである.

- このタイプの解法は, Inexact Newton Method あるいは pseudo-Newton Method という形で多変数に拡張される. 今日では大域的に収束する解法が知られていることも Newton 法と同じである.

- 非線形方程式

$$e^x - 10 = 0$$

を Newton 法およびセカント法によって解くプログラム例とその実行結果を次ページに示す. ただし, 初期値を 1.5, 終了条件を

$$|e^x - 10| < 10^{-3}$$

としている. また, セカント法では, 繰り返し

の最初の 1 回のみ $f'(x)$ を用いている.

Newton 法のプログラム例

```
deff('y=f(x)','y=exp(x)-10');
deff('y=df(x)','y=exp(x)');
err=1.0E-3;x=1.5;k=1;v=f(x);
while(1)
  if(abs(v)<err)
    break;
  else
    dv=df(x);
    x=x-v/dv;
    v=f(x);
    k=k+1;
  end
end
```

Newton 法の $f(x)$ の履歴

繰り返し回数	$f(x)$
1	-5.518311
2	5.352857
3	0.833532
4	0.031259
5	0.000049

セカント法のプログラム例

```
deff('y=f(x)','y=exp(x)-10');
deff('y=df(x)','y=exp(x)');
err=1.0E-3;x=1.5;k=1;v=f(x);
while(1)
  if(abs(v)<err)
    break;
  else
    if(k==1)
      dv=df(x);
    else
      dv=(v-v0)/(x-x0);
    end
    x0=x;v0=v;
    x=x-v/dv;v=f(x);k=k+1;
  end
end
```

セカント法の $f(x)$ の履歴

繰り返し回数	$f(x)$
1	-5.518311
2	5.352857
3	-1.626929
4	-0.355964
5	0.033399
6	-0.000608

同一の終了条件のもとで, 計算終了までに要した繰り返しの回数は, この例では, 以下の表の通りであった.

2 分法	13 回
はさみうち法	8 回
Newton 法	5 回
セカント法	6 回

高次解法

- Newton 法はテイラー展開の 1 次の項までを使う.
- これに対し, 高次解法とは, テイラー展開のより高次の項を使う解法の総称である.

- この種の解法は, 高次の項を使えばより高速な解法が得られるかもしれないという期待に基づいて作られているのだが, 実用上は必ずしもそうなるわけではない.
- このため, 高次解法は, 高速化のためというよりは, 特別な目的がある場合に使用される.

- 次回の講義で紹介する平野法は高次解法の一種で, 代数方程式の重根が求められるよう工夫された手法である.
- $f(x)$ が多項式で, a が $f(x) = 0$ の重根であるとき, $f'(x) = 0$ となるので, 求解に単純な Newton 法を適用することはできない. 平野法はこの問題を解決するために作られた方法であり, 目的は高速化ではない.

- このような特別な場合を除き, 高次解法はあまり使われないので, この講義ではこれ以上の説明を省略する.

複素解

- これまでの議論では実数解のみを対象にしてきた.
- 複素解を求めることも非線形方程式を解くことの一種なのであるが, 慣例的に, 非線形方程式の複素解を求める方法は, 代数方程式の解法と呼ばれることがある.

- 代数方程式の解法については次回に述べるが, 複素数を取り扱うことができる処理系でプログラムを組む場合, 実根を求めるための Newton 法と複素根を求めるための Newton 法には特に相異はない. ただし, 重根の場合には, 先に平野法の説明で述べたように, 何らかの工夫が必要となる.

多変数の Newton 法

- $\boldsymbol{x}$ を n 次のベクトル, $\boldsymbol{f}(\boldsymbol{x})$ を n 次のベクトル値関数とする.
- 多変数の非線形方程式を解くということは, $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ という非線形連立方程式を解くということである ($\boldsymbol{0}$ は零ベクトル).

- 以下では,

$$\boldsymbol{J}(\boldsymbol{x}_k) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_k}$$

とする.

- 多変数の Newton 法の考え方は,

$$\boldsymbol{f}(\boldsymbol{x}_k + \boldsymbol{d}) \simeq \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d}$$

と近似し, 本来解くべき方程式

$$\boldsymbol{f}(\boldsymbol{x}_k + \boldsymbol{d}) = \boldsymbol{0}$$

を

$$\boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d} = \boldsymbol{0}$$

で置き換えるというものである.

- $\boldsymbol{d}_k$ を変数としたとき, 方程式

$$\boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d}_k = \boldsymbol{0}$$

の解は, 連立 1 次方程式

$$\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d}_k = -\boldsymbol{f}(\boldsymbol{x}_k)$$

の解である.

- 先の $\boldsymbol{d}_k$ を用いて,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_k$$

という繰り返し計算をおこなうのが, 多変数 Newton 法の基本的な考え方である.

- アルゴリズムの形で書き下すと, 次ページ以降のようになる.

▷ **初期化:** 初期値 $\boldsymbol{x}_0$ と誤差の許容値 $\varepsilon > 0$ を定める. $k = 0$ とする.

▷ **ループ:** $|\boldsymbol{f}(\boldsymbol{x}_k)| < \varepsilon$ であれば終了. そうでない場合には,

$$\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d} = -\boldsymbol{f}(\boldsymbol{x}_k)$$

という線形方程式を解いて d を求め,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}$$

とし, k に 1 を加えてループの先頭に戻る.

- 1 変数の Newton 法では f の導関数 $f'(x_k)$ であった部分が, 多変数の Newton 法でははは $\boldsymbol{f}$ の Jacobi 行列 $\boldsymbol{J}(\boldsymbol{x}_k)$ に変わっている.

- 1 変数の Newton 法における

$$\frac{1}{f'}$$

に対応するのが, 多変数の Newton 法における

$$(\boldsymbol{J}(\boldsymbol{x}_k))^{-1}$$

(逆行列) であるが…

- 数値的には, 逆行列を直接求めるのではなく, 線形方程式

$$\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{d} = -\boldsymbol{f}(\boldsymbol{x}_k)$$

を解く.

- 教科書には2変数の場合のみ書かれているが, 変数がいくつあってもやることは同じである.
- 多変数の Newton 法は, Jacobi 行列 $\boldsymbol{J}(\boldsymbol{x}_k)$ が正則でない場合には使えない.

- 単純な Newton 法が収束するか否かは初期値に依存する．直線探索法や信頼領域法を使えば初期値によらず収束するアルゴリズムが作れることも，1 変数の Newton 法と同様である．

- 多変数の問題では, Jacobi 行列を求めるだけでも CPU の負荷が高すぎる場合がある. このような場合には, Jacobi 行列の近似が用いられる (1 変数のセカント法に対応).
- 近似の方法には様々なものがあり, 今日では大域的に収束する手法が知られている.

Newton 法で関数の最小値を求める

- 関数 $f(\boldsymbol{x})$ が最小になる点 x を求める問題を考える.
- 最大値を求める問題も同様に取り扱える.
- $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ は n 次のベクトルで, f は実数値関数とする.

- f は 2 階連続微分可能と仮定する.
- $\boldsymbol{p}(\boldsymbol{x}) = \left(\dfrac{\partial f}{\partial x_1}, \ldots, \dfrac{\partial f}{\partial x_n} \right)$ とする.

- $\boldsymbol{H}(\boldsymbol{x}) = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 x_n} \\ \vdots & & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$ とする.

- 行列 $\boldsymbol{H}$ が x_* で正値なら f は x_* で最小値となり, $\boldsymbol{H}$ が x_* で負値なら f は x_* で最大値となることが示せる.
- 以下の解説では f が最小値を持つことが前提であるので, $\boldsymbol{H}(\boldsymbol{x})$ は $\boldsymbol{x}$ を固定すると正定となることを仮定する.

- 初期値を $\boldsymbol{x}_0$ とし, $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{d}$ とした場合 (すなわち解をベクトル $\boldsymbol{d}$ の分だけ動かした場合) の関数 f の値の変動を調べる.
- 記法の簡単のため, $\boldsymbol{p}_0 = \boldsymbol{p}(\boldsymbol{x}_0)$, $\boldsymbol{H}_0 = \boldsymbol{H}(\boldsymbol{x}_0)$ と書く.

- Taylor 展開の 2 次の項まで取って近似すると，次式が得られる．

$$
\begin{aligned}
f(\boldsymbol{x}) &\simeq f(\boldsymbol{x}_0) + \boldsymbol{p}_0\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T\boldsymbol{H}_0\boldsymbol{d} \\
&= f(\boldsymbol{x}_0) - \frac{1}{2}\boldsymbol{p}_0^T\boldsymbol{H}_0^{-1}\boldsymbol{p}_0 \\
&\quad + \frac{1}{2}(\boldsymbol{d} + \boldsymbol{H}_0^{-1}\boldsymbol{p}_0)^T\boldsymbol{H}_0(\boldsymbol{d} + \boldsymbol{H}_0^{-1}\boldsymbol{p}_0)
\end{aligned}
$$

- $\boldsymbol{H}_0$ は正定であると仮定したから, $f(\boldsymbol{x})$ の近似値は $\boldsymbol{d} = -\boldsymbol{H}_0^{-1}\boldsymbol{p}_0$ としたとき最小になる.
- Newton 法で関数の最小値を求めるアルゴリズムは, この考え方を用いて繰り返し計算を実行する.

- 記法の簡単のため,

$$\boldsymbol{p}_k = \boldsymbol{p}(\boldsymbol{x}_k)$$
$$\boldsymbol{H}_k = \boldsymbol{H}(\boldsymbol{x}_k)$$

と書く.

- 非線形 (多変数) 最小化問題を解くための Newton 法の基本形は以下の通りである.

▷ 初期化: 初期値 $\boldsymbol{x}_0$ と誤差の許容値 $\varepsilon > 0$ を定める. $k = 0$ とする.

▷ ループ: $|\boldsymbol{p}(\boldsymbol{x}_k)| < \varepsilon$ であれば終了．そうでない場合には,

$$\boldsymbol{H}_k \boldsymbol{d} = -\boldsymbol{p}_k$$

という線形方程式を解いて $\boldsymbol{d}$ を求め,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}$$

とし, k に 1 を加えてループの先頭に戻る.

- 終了条件には色々な取り方がある.
- 上記では $\boldsymbol{p}(\boldsymbol{x}_k)$ のノルムが一定以下になったら終了としているが, これは最小値に近付くほど接線の傾きが水平に近付くという性質を使っている.

- 収束が局所的であることは非線形方程式に対する Newton 法と同じ.
- 最小化したい関数が 2 次関数なら, 1 回の計算で最小値が得られる.
- 直線探索や信頼領域法を用いることで大域的な収束解法が得られることも今までと同様である.

- 非線形方程式 $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ を解く問題を $\boldsymbol{f}^T(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$ の最小値を求める問題に書き直してから解くことがある.
- 数値計算の誤差に対応するためにはこの方が有利なこともある.

Scilab で非線形方程式を解く

- Scilab で非線形方程式を解くには，fsolve という関数を使う．
- アルゴリズムは修正 Powell 混合法である (典拠はオンラインマニュアル参照)．

- 次ページに

$$x^3 - 2 = 0$$

を fsolve で解くプログラムおよび実行結果を示す (教科書 6 ページも参照).

- プログラム:

```
deff('y=f(x)','y=x^3-2');
x0=1; //初期値
fsolve(x0,f)
```

- 実行結果:

```
ans   =
1.2599210
```

- 関数定義について注意しておくと, 関数名は, 組み込み関数や予約語と衝突しなければ, 何でもよい.

- Scilab では, 特に指定しなければ, ans という変数に計算結果が格納される. この例では数値解は 1.2599210 で, 厳密解 $\sqrt[3]{2}$ と浮動小数点数の誤差のレベルで一致する.

- fsolve には初期値依存性があるので注意. 特に解が複数ある場合は要注意.
- fsolve は解きたい関数の Jacobian を与えても与えなくても解けるが, 一般に Jacobian を与えた方が高速.

Scilab で非線形最適化問題を解く

- 組み込み関数 optim か fminsearch を使う.
- optim および fminsearch 解くのは制約条件なしの最小化問題である. 関数 f を最大化したいときには, かわりに関数 $-f$ を最小化すればよい.

- optim を使う場合には, 最小化したい関数に加えて, その勾配ベクトルと終了フラグを与える必要がある.
- 次ページに

$$f(x) = x^2 + x + 1$$

を最小化するプログラムと実行結果を示す.

- プログラム:

```
function [f,j,ind]=costFn(x,ind)
  f=x^2+x+1;
  j=2*x+1;
endfunction

x0=0;
[fopt,xopt]=optim(costFn,x0);
```

- 実行結果

```
--> fopt
 fopt  =
   0.75
--> xopt
 xopt  =
  -0.5
```

- 先のプログラムにおいて, f が最小化したい関数, j がその導関数になっている.

- 最小化したい関数は

$$f(x) = \left(x + \frac{1}{2}\right)^2 + \frac{3}{4}$$

なので, $x = -1/2$ で最小値 $3/4$ を取る. 数値計算の結果は, 確かにこれと一致する.

- 続いて，fminsearch を用いて同じ問題を解いたプログラム例および実行結果を示す．

- プログラム:

```
deff('y=f(x)','y=x^2+x+1');
x0=0;
[xopt,fopt,flag]=fminsearch(f,x0);
```

- 実行結果

```
--> xopt
 xopt  =
  -0.4999805
--> fopt
 fopt  =
   0.7500000
```

- この例では, optim と比較して解に含まれる誤差が大きくなっている.