10/31, 11/28 休, 1/31 試.

(10/3)

IEEE 標準の浮動小数点数(1)

- 倍精度の場合(単精度はお勧めしない)
  - 64 bit: 符号部 1 bit, 指数部 11 bit, 仮数部 52 bit
  - 値: $y = (-1)^s 2^e f$

s	e'	f

  - 指数部 e は biased exponent ($e = e' - 1023$)
    - 正規化数の指数範囲 $-1022 \leqq e \leqq 1023$
  - 仮数部 f は頭の1を省略 ($f = 1.f'$) 実質 53 bit
    - マシンイプシロン ε: $\varepsilon > 0$, $fl(1.0 + \varepsilon) = 1.0 + \varepsilon$
    - 丸め単位 $u = \varepsilon/2$: $fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta)$ with $|\delta| < u$
    - $\varepsilon = 2^{-52} \fallingdotseq 2.22 \times 10^{-16}$, $u = 2^{-53} \fallingdotseq 1.11 \times 10^{-16}$

単精度:指数部 8 bit ($e = e' - 127$), 仮数部 23 bit

1

IEEE 標準の浮動小数点数(2)

- 絶対値が小さくなると・・・ underflow
  - 非正規化数:指数部 $e' = 0$ のとき, $f = 0.f'$ とする
    - 相対精度が低下する
  - 0:すべてのビットが 0 のときだが, 符号はつけられる
- 絶対値が大きくなると・・・ overflow
  - 無限大:やはり符号付き
- 特殊な数
  - NaN (Not a Number): 0/0, sqrt(-1) など

2

二次方程式の解の公式(1)

- $0.02\,x^2 + 802\,x + 1.01 = 0$ を, 公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

  を使って求める
- 倍精度:-4.0099998e+4 & -1.2593517e-3
- 単精度:-4.0100000e+4 & -9.7412721e-4
- 相対精度が恐ろしく悪い
  - 単精度がいかに使えないか・・・

3

倍精度が正しいとは
限らない

二次方程式の解の公式(2)

- 単精度と倍精度の計算を逐一比較すると

	単精度	倍精度
b	8.02	8.02
sqrt(d)	8.01999961	8.01999950
-b + sqrt(d)	0.00000039	0.00000050

- ほぼ同じ大きさの 2 つの数の引き算において, 相対精度が極端に下がる

  「桁落ち」(cancellation) という

4

無限和の近似計算(1)

- 無限和

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \approx 1.64493407$$

  を単精度で計算すると, $k = 4096$ で変化がなくなり, 和の値は1.64472532
- 有効数字が 4 桁しかない!
- 大きい「部分和」に小さい「項」が足されるときに, 情報落ちが発生している

Nicholas J. Higham: Accuracy and Stability of Numerical Algorithms, SIAM

5

大+小 → 情報落ち

無限和の近似計算(2)

- 倍精度で計算すると 1.64469002
- 単精度で k = 1, 2, …, 4069 の順: 1.64472532
- 単精度で k = 4096, 4095 … の順: 1.64469004
  - 逆順に計算した 1.64469002 すら

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi}{6} \approx 1.64493407$$

  にはあまり近くない
    - 変化がなくなっても, 真値に近いとは限らない!
    - 真値に近いかどうかは, 別の方法で確認すべし

6

数値微分(1)

- 関数 $f(x)$ の微分の計算
  - 原理は簡単だが，関数が複雑だと長い式になる
    - 当然計算量が増える
- 微分の「定義」に戻って近似する

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

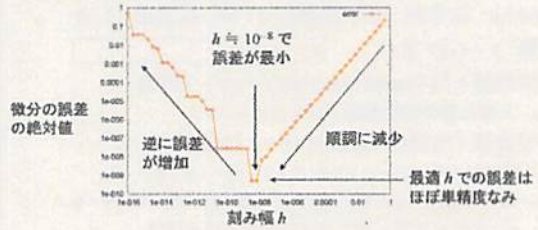
「数値微分」と呼ぶ

- たった 2 倍の計算量；デバッグにも使える
- h はどうしたらよいか？

7

数値微分(2)

- 実験：$f(x) = \sin(x)$ で，$x = 1.0$ とした



8

一読の価値がある参考書

- 伊理正夫・藤野和建：数値計算の常識，共立
- 二宮市三ほか：数値計算のつぼ，共立
  - いずれも，アルゴリズム以前に重要な，数値計算の【センス】や【考え方】に詳しい
  - アルゴリズムを集める(知っている)だけでは，適切な数値計算はできない！

9

浮動小数の誤差モデル

Nicholas J. Higham: Accuracy and Stability of Numerical Algorithms, SIAM

- 四則演算と平方根に対して

$$\mathrm{fl}(x \mathbin{\mathrm{op}} y) = (x \mathbin{\mathrm{op}} y)(1+\delta) \quad |\delta| \leqq u = 2^{-53}$$

$u = \sup\{\delta\}$ ではない

- つぎのようにも書ける

$$\mathrm{fl}(x \mathbin{\mathrm{op}} y) = \frac{x \mathbin{\mathrm{op}} y}{1+\delta}$$

- さらに，$|\delta_i| \leqq u$, $\rho_i = \pm 1$, $mu < 1$ のとき

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n \quad |\theta_n| \leq \frac{mu}{1-mu} =: \gamma_n$$

帰納法で丁寧に証明する

10

和・積を和に変換するアルゴリズム

- Møller-Knuth のアルゴリズム
  - $z = x + y;\ w = z - x;\ zz = (y - w) - ((z - w) - x)$
  - $x + y = z + zz$ が厳密に成り立つ
- Dekker のアルゴリズム　Numerische Mathematik, 18, 224-242 (1971)
  - $p = x * c;\ hx = (x - p) + p;\ tx = x - hx;$
    $p = y * c;\ hy = (y - p) + p;\ ty = y - hy;$
    $p = hx * hy;\ q = hx * ty + tx * hy;$
    $z = p + q;\ zz = ((p - z) + q) + tx * ty$
  - $x * y = z + zz$ が厳密に成り立つ

hx は x の上 26 ビット(27 ビット目を四捨五入)

$c = 2^{27} + 1$

11

区間演算

- 数 x の近似を，区間で表す　$x \in [\underline{x}, \overline{x}]$
- 区間どうしの演算を定義

$$[\underline{x},\overline{x}]+[\underline{y},\overline{y}] = [\underline{x}+\underline{y},\overline{x}+\overline{y}]$$

$$[\underline{x},\overline{x}]-[\underline{y},\overline{y}] = [\underline{x}-\overline{y},\overline{x}-\underline{y}]$$

$$[\underline{x},\overline{x}]*[\underline{y},\overline{y}] = [\min\{\underline{x}\underline{y},\underline{x}\overline{y},\overline{x}\underline{y},\overline{x}\overline{y}\},\max\{\underline{x}\underline{y},\underline{x}\overline{y},\overline{x}\underline{y},\overline{x}\overline{y}\}]$$

$$[\underline{x},\overline{x}]/[\underline{y},\overline{y}] = [\underline{x},\overline{x}]*[1/\overline{y},1/\underline{y}]$$

- これは厳密な計算での話
- 実際には、浮動小数の丸めも考慮する

12

連続系アルゴリズム レポート課題3

- 宿題(レポート)
  - 提出状況に応じて，期末試験の点に最大 50 点を加算します
    - まったく提出しなくても，期末試験の点で成績が出ます
    - ただし，試験を受けなければ単位は出ません
  - プログラムだけはメールの添付ファイルで
    reiji@is.s.u-tokyo.ac.jp
    に送付してください
    - 所属学科，学年，学籍番号，氏名をメール本文に明記すること
    - メールの題名(Subject)は「連続系レポート課題第3回 提出者氏名」とすること
  - 手計算をするものや，計算結果と考察など，プログラム以外は A4 の紙 1 枚(裏もつかってよい)にまとめてください
    - 所属学科，学年，学籍番号，氏名をレポートの最初に明記すること
    - 次回の講義の前に集めます
    - あるいは PDF で A4 サイズ 2 ページでもよい(プログラムと一緒にメールで送付してください)

- 問題1
  - a を実数定数として，以下の関数 f(x) に対して，f(x) = 0 の根を求める Newton 法の式を導け．
    (1) $f(x) = \frac{1}{x} - a$
    (2) $f(x) = \frac{1}{x^2} - a$

- 問題2
  - Newton 法により次の値を求めるプログラムを書け
    (1) [1, 2] の実数 x に対する逆数 1/x
    (2) [1, 4] の実数 x に対する平方根の逆数 $x^{-1/2}$
  - ただし，使う浮動小数演算は加減算と乗算のみとする．反復回数は固定でもよい．
    - いくつか値を与えて，精度を調べよ

おまけ

- $z^3 - 1 = 0$ の根を求める複素 Newton 法のプログラムを書け
- $z = x + iy$ ($-1 \leq x \leq 1, -1 \leq y \leq 1$)を初期値として上記のプログラムの反復回数を得よ
- z に対して反復回数をプロットせよ

C 言語補足

- printf の書式いろいろ
  - 科学技術形式

```
printf("%e¥n", x);
```

    - 3.1415e+00 のような出力になる(指数部の桁数はシステムや表現する数によって 3 桁のことも)
    - これは 3.1415×10^0 の意味
    - 入力でも使える: 3e+12 とか
  - 有効数字の設定

```
printf("%14.6e¥n", x);
```

    - 全体で 14 文字、小数点以下 6 桁
    - 3.141592e+00 のような出力になる
    - 符号、整数部、小数点、指数部の文字数をよく数えて、十分な文字数を配置する
      - 足りない時にはいろんなことがおきる
  - 整数を出力

```
printf("%8d¥n", i);
```

    - これだと 8 桁で表示

- 関数
  - 似たような計算を繰り返す時には関数を使う
  - まずは関数の「宣言」

```
double y(double x) {
  return 3.0 * x + 1.0;
}
```

    - 最初の double は「返り値」の型
    - つぎの y は関数の名前
    - つづく丸カッコ内 double x は「引数」の型と名前
      - これは関数本体で変数同様に扱われる
    - 最後の波カッコ内は関数の「本体」(計算内容)
      - ここでは return 文しかない
    - return 文では、関数が返す値を示すとともに、そこで関数の計算が終わることを意味する
  - 次に関数の呼び出し
    - 別の関数などにおいて式の一部として書ける

```
dy = (y(1.1) – y(1.0)) / 0.1;
```

    - これは ((3.0 * 1.1 + 1.0) – (3.0 * 1.0 + 1.0)) / 0.1 が dy に代入されることになる

関数のつづき

- 関数内で、引数は変数と同じ扱い
  - 代入もできるが、呼び出し元に影響はない

```
double f(double x, double y) {
  x *= y;
  return x * x;
}

int main(void) {
  double a = 2.0, b = 2.0, c;
  c = f(a, b);
  printf("%f %f %f¥n", a, b, c);
  return 0;
}
```

  - 出力は 2.00 2.00 16.00 である
    - つまり、main の中の a は変更されない
  - 関数 f を実行する際には、main の中の a の値が、f の中の x にコピーされる
- おまけ
  - 関数 f の宣言は、引数が複数ある例にもなっている
  - x *= y は x = x * y と同じこと
  - main にあるように変数の宣言時に初期値を指定できる
    - 初期値を指定しないと、滅茶苦茶な値が入っているので、使うときにはまっとうな値を代入してやらないといけない

- 関数ごとに変数は別物

```
void f(void) {
  int a;
  a = 1;
}

int main(void) {
  int a = 0;
  f();
  return a;
}
```

  - f は引数のない関数、返り値のない関数の例
    - 呼び出しかたも確認してほしい(fortran と違う)
  - main 内でも f 内でも、いずれも a という変数が宣言されているが、これらは「別物」
    - それぞれに違う値をしまっておける
  - 上記の例では main は 0 を返す(1 ではない)

制御構文一式

- 文
  - C 言語における実行の単位
  - 例:代入文　　x=0;
  - 例:関数呼び出し　printf(ほにゃらら);
  - 例:空文　　;
- 複文
  - { と } の間に、複数の文がある
  - { から } までの中身が、1 つの文と同等になる
- if 文
  - 書き方1: if(条件) 文
  - 書き方2: if(条件) 文 else 文
  - 入れ子に注意

```
if(条件1)
  if(条件2)
    文1
  else
    文2
```

```
if(条件1) {
  if(条件2)
    文1
} else
  文2
```

- 条件(a <= 0 のたぐい)
  - C 言語の条件は、0 が偽で、それ以外が真
  - if(1) 文1 else 文2 は、常に文1 が実行される

- while 文
  - 書き方: while(式) 文
  - 「式」が成り立っている限り、「文」が繰り返される
  - do 文というのもある
- 無限ループ
  - while(1) 文
  - for(;;) 文
- break 文
  - 書き方: break;
  - 最内側 for ループや while ループから抜け出す

```
while(1) {
  前半の計算
  if(終了条件)
    break;
  後半の計算
}
```

```
for(c=0; c< MAX; c++) {
  前半の計算
  if(終了条件)
    break;
  後半の計算
}
```

- Continue 文
  - 最内側 for ループや while ループの、現在の反復の残りの処理をスキップして、次の反復に移る

m 重根の場合の収束速度

- 同様に Taylor 展開：重根より $f(x^*) = f'(x^*) = 0$

$$f(x) = f(x^*) + (x-x^*)^m f^{(m)}(x^*)/m! + (x-x^*)^{m+1} R$$

$$f'(x) = f'(x^*) + (x-x^*)^{m-1} f^{(m)}(x^*)/(m-1)! + (x-x^*)^m R'$$

$$x - \frac{f(x)}{f'(x)} - x^* = x - x^* - \frac{(x-x^*)^m f^{(m)}(x^*)/m! + (x-x^*)^{m+1} R}{(x-x^*)^{m-1} f^{(m)}(x^*)/(m-1)! + (x-x^*)^m R'}$$

$$= x - x^* - (x-x^*)\frac{1}{m}\frac{f^{(m)}(x^*) + r}{f^{(m)}(x^*) + r'} \qquad \frac{f^{(m)}(x^*)+r'-r'+r}{f^{(m)}(x^*)+r'}$$

$$= \left(1 - \frac{1}{m}\right)(x-x^*) + (x-x^*)^2 \frac{1}{m}\frac{r'-r}{f^{(m)}(x^*)+r'}$$

これは 1 次収束：しかし更新量を m 倍すれば 2 次収束になる

1

Newton 法の減速

- Newton 法は遠くの解に飛んでいってしまうことがある
- 遠くに行かないように「減速」すると有効なことがある
- 減速 Newton 法
  - 適当なパラメタ $0 < \beta < 1$ と $\lambda > 1$ を決めておく
  - もとの Newton 法 1 ステップを $x^{(k+1)} = x^{(k)} + d^{(k)}$ として

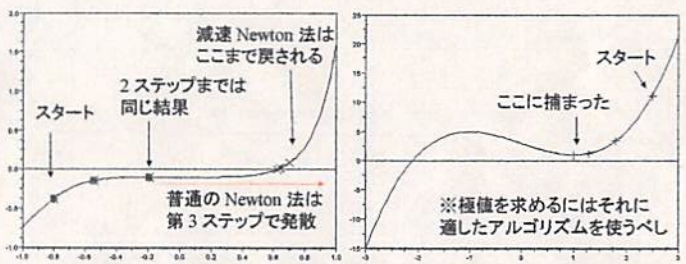
    $\| f(x^{(k)} + \lambda^{-j} d^{(k)}) \| \leqq (1 - (1-\beta)\lambda^{-j}) \| f(x^{(k)}) \|$

    となる最小の正整数 j を求め，$x^{(k+1)} = x^{(k)} + \lambda^{-j} d^{(k)}$
    - 条件を満たす j が必ず存在し，$J(x^*)$ が正則なら x^* の十分近くで $j = 0$ となる（標準 Newton 法と一致する）ことが言える

2

減速 Newton 法の効果と限界

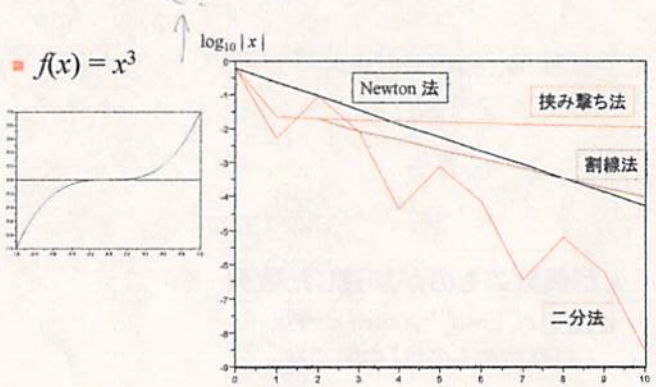
- $f(x) = \exp(x^5) - 1.1$
- $f(x) = x^3 - 3x + 3$



3

二分法のロバスト性（2）

- $f(x) = x^3$



4

Richardson 加速

- 数値微分 $a_n = [\sin(1 + 2^{-n}) - \sin(1)] / 2^{-n}$

1	0.5	0.3120480035933				cos(1.0) =	0.5403023058681
2	0.25	0.4300545381900	0.5480610727892				
3	0.125	0.4883728743296	0.5425912104684	0.5409012583615			
4	0.0625	0.5138632057465	0.5409535371640	0.5402743127293	0.5402980350674		
5	0.03125	0.5270874581480	0.5404717065480	0.5403110963410	0.5403020654284	0.5403022874525	
6	0.015625	0.5337064828577	0.5403454689608	0.5403033905759	0.5403022897524	0.5403023047073	0.5403023059091
7	0.007813	0.5370058303297	0.5403131978017	0.5403024405481	0.5403023048275	0.5403023058326	0.5403023058609
8	0.003906	0.5386574356820	0.5403050414342	0.5403023226451	0.5403023058021	0.5403023058871	0.5403023058682
9	0.001953	0.5394830138059	0.5403029912296	0.5403023079618	0.5403023058640	0.5403023058681	0.5403023058691
10	0.000977	0.5398913455177	0.5403024774296	0.5403023061295	0.5403023058676	0.5403023058680	0.5403023058680
11	0.000488	0.5400965471504	0.5403023487530	0.5403023059000	0.5403023058682	0.5403023058682	0.5403023058682
12	0.000244	0.5401995818752	0.5403023168000	0.5403023058733	0.5403023058682	0.5403023058682	0.5403023058682
13	0.000122	0.5402508452133	0.5403023085014	0.5403023058696	0.5403023058690	0.5403023058680	0.5403023058690
14	6.1E-05	0.5402765258756	0.5403023065383	0.5403023058672	0.5403023058670	0.5403023058669	0.5403023058669
15	3.05E-05	0.5402894659584	0.5403023060371	0.5403023058701	0.5403023058705	0.5403023058707	0.5403023058708
16	1.53E-05	0.5402958858040	0.5403023059116	0.5403023058698	0.5403023058697	0.5403023058697	0.5403023058696
17	7.63E-06	0.5402990958119	0.5403023058898	0.5403023058825	0.5403023058843	0.5403023058853	0.5403023058855
18	3.81E-06	0.5403007006008	0.5403023058898	0.5403023058896	0.5403023058908	0.5403023058913	0.5403023058914
19	1.91E-06	0.5403015033506	0.5403023058467	0.5403023058510	0.5403023058454	0.5403023058424	0.5403023058406
20	9.54E-07	0.5403019046844	0.5403023059480	0.5403023059771	0.5403023059951	0.5403023060051	0.5403023060103
21	4.77E-07	0.5403021053450	0.5403023058316	0.5403023057928	0.5403023057684	0.5403023057512	0.5403023057430
22	2.38E-07	0.5403022053652	0.5403023054023	0.5403023053659	0.5403023053049	0.5403023052742	0.5403023052588
23	1.19E-07	0.5403022561222	0.5403023065793	0.5403023073456	0.5403023076277	0.5403023077625	0.5403023078635
24	5.96E-08	0.5403022803366	0.5403023045510	0.5403023037749	0.5403023032649	0.5403023029740	0.5403023028199
25	2.98E-08	0.5403022952378	0.5403023101389	0.5403023120016	0.5403023131788	0.5403023138378	0.5403023141881
26	1.49E-08	0.5403022989631	0.5403023026884	0.5403023002048	0.5403022985196	0.5403022975424	0.5403022970188
27	7.45E-09	0.5403023064137	0.5403023138842	0.5403023175895	0.5403023200730	0.5403023215099	0.5403023222531
28	3.73E-09	0.5403023064137	0.5403023064137	0.5403023038301	0.5403023019780	0.5403023007725	0.5403023001935

まったく収束しない。

丸め

$a_n^{(1)}$

$a_n^{(2)}$

Aitken 加速

伊理・藤野「数値計算の常識」p.145

表 20-3 数列 $a_n = \sum_{k=0}^{n} \frac{(-1)^k}{2k+1} \to \frac{\pi}{4} = 0.78539816\cdots$ の Aitken 加速

n	a_n	$a_n^{(1)}$	$a_n^{(2)}$	$a_n^{(3)}$	$a_n^{(4)}$
0	1.000000D+00				
1	6.666667D-01				
2	8.666667D-01	7.916667D-01			
3	7.238095D-01	7.833333D-01			
4	8.349206D-01	7.863095D-01	7.855263D-01		
5	7.440115D-01	7.849206D-01	7.853626D-01		
6	8.209346D-01	7.856782D-01	7.854108D-01	7.853998D-01	
7	7.542680D-01	7.852203D-01	7.853926D-01	7.853977D-01	
8	8.130915D-01	7.855180D-01	7.854007D-01	7.853983D-01	7.853982D-01
9	7.604599D-01	7.853137D-01	7.853968D-01	7.853981D-01	7.853982D-01
10	8.080730D-01	7.854599D-01	7.853989D-01	7.853982D-01	7.853982D-01

Aitken 加速の結果は表 20-3 のとおりで，$\pi/4$ への収束が大変に遅いとされている数列であるにもかかわらず，初めの 10 項くらいの情報から 7~8 桁正しい極限値が得られてしまう．

ε アルゴリズム

- やはり $a_n = a + c_1\lambda_1^n + c_2\lambda_2^n + \cdots$

$$\begin{cases} \varepsilon_n^{(-1)} = 0 \\ \varepsilon_n^{(0)} = a_n \\ \varepsilon_n^{(k)} = \varepsilon_{n-1}^{(k-2)} + \dfrac{1}{\varepsilon_n^{(k-1)} - \varepsilon_{n-1}^{(k-1)}} \end{cases}$$

- k が偶数のものが加速した数列
  - 特に $\varepsilon_n^{(2)} = a_n^{(1)}$ Aitken と一致
  - k が奇数のものは「中間変数」

ε 加速

伊理・藤野「数値計算の常識」p.146

$$P = \begin{bmatrix} 0.1 & 0.1 & 0.8 \\ 0.8 & 0.1 & 0.1 \\ 0.1 & 0.8 & 0.1 \end{bmatrix}, \quad x^{(0)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

として，P を $x^{(0)}$ に次々と掛けて得られるベクトル

$$x^{(n)} = P^n \cdot x^{(0)}$$

の第1成分の作りだす数列 a_n は，

$$\left.\begin{aligned} a_n &= \frac{1}{3} + (0.7)^n\left(c_1\cos\left(\frac{2n\pi}{3}\right) + c_2\sin\left(\frac{2n\pi}{3}\right)\right) = \frac{1}{3} + b_1\lambda_1^n + b_2\lambda_2^n, \\ |\lambda_1| &= |\lambda_2| = 0.7, \quad \lambda_1 = 0.7\cdot\exp\left(\frac{2\pi i}{3}\right), \quad \lambda_2 = \bar{\lambda}_1 \end{aligned}\right\} \quad (6)$$

表 20-5 式(6)の数列に対する ε 算法の適用

n	$\varepsilon_n^{(0)} = a_n$	$\varepsilon_n^{(1)}$	$\varepsilon_n^{(2)} = a_n^{(1)}$	$\varepsilon_n^{(3)}$	$\varepsilon_n^{(4)}$
0	1.000000D+00				
1	1.000000D-01	-1.111111D+00			
2	1.700000D-01	1.428571D+01	1.649485D-01		
3	5.620000D-01	2.551020D+00	8.478261D-02	1.811574D+00	
4	2.533000D-01	-3.239391D+00	3.893007D-01	5.834898D+00	3.333333D-01
5	2.773100D-01	4.184931D+01	2.755773D-01	-1.203266D+01	3.333333D-01
6	4.117660D-01	7.437377D+00	2.480804D-01	5.281557D+00	3.333333D-01
7	3.058819D-01	-9.444289D+00	3.525301D-01	1.701136D+01	3.333333D-01
8	3.141173D-01	1.214266D+02	3.135230D-01	-3.598063D+01	3.333333D-01

数列の収束

$a_1, a_2, a_3, \cdots$ が a に収束するとする

1次収束：十分大きい n に対し，

$0 < M < 1$ なる M があり，

$$|a_{n+1} - a| \leq M |a_n - a|$$

p次収束 $(p > 1)$

十分大きい n に対し，$0 < M$ があって，

$$|a_{n+1} - a| \leq M |a_n - a|^p$$

非線形方程式

1変数の実数な関数 $f(x)$

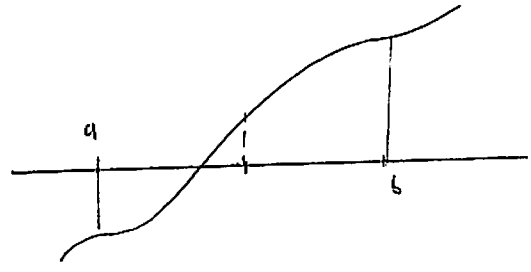
$f(x) = 0$ をみたす．

二分法

$f(a) f(b) < 0$

$a < x < b$ なる $f(x) = 0$ が

1つ以上ある．



$c := \frac{a+b}{2}$ とする

$f(a) f(c) < 0$ なら $b := c$

else $a := c$

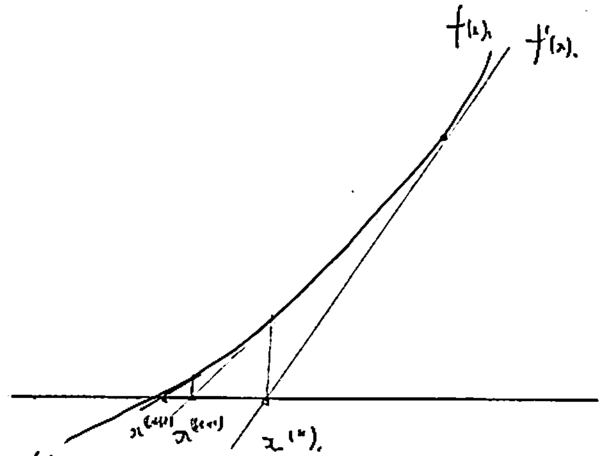
⇒ a と b が浮動小数で表現できる最小の差になったら終了

$M=\frac{1}{2}$ の 1次収束.

Newton (-Raphson) 法.

方程式 $f(x)=0$

1回連続微分可能



Taylor展開. 反復回数.

$$f(x^{(k+1)}) \fallingdotseq f(x^{(k)}) + (x^{(k+1)} - x^{(k)}) f'(x^{(k)})$$

$= 0$ とおく.

これより.

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

を繰り返す.

多変数 ver もつくれる
立つかも

単根の場合, 根を x^* として,

$$f(x^*) = f(x) + (x^* - x) f'(x) + \frac{(x^* - x)^2}{2} f''(\xi) = 0.$$

$$x^* - x^{(k+1)} = x^* - x^{(k)} + \frac{f(x^{(k)})}{f'(x^{(k)})}$$

$$= x^* - x^{(k)} - \frac{(x^* - x^{(k)}) f'(x^{(k)}) + \frac{(x^* - x)^2}{2} f''(\xi)}{f'(x^{(k)})}$$

$$= -\frac{(x^* - x^{(k)})^2}{2} \frac{f''(\xi)}{f'(x^{(k)})}$$ 単根より $f' \neq 0$.

これより, 2次収束であることがわかる.

(単根から つながる)

Newton法の正しい止め方.

$|f(x^{(k)})| \leq |f(x^{(k+1)})|$ となったら

止める.

m 重根のときは

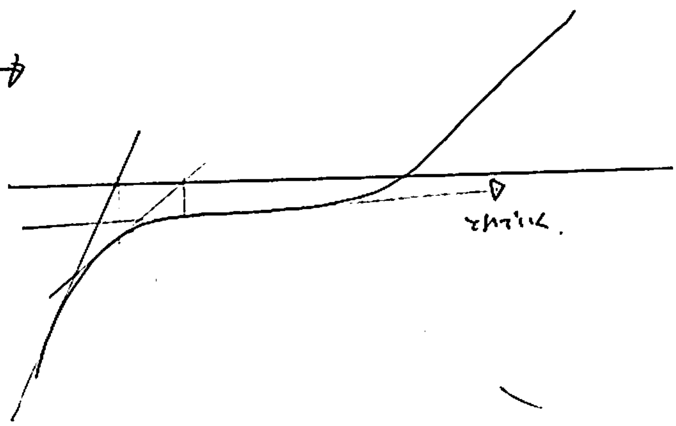
$$M = 1 - \frac{1}{m}$$

の 1次収束となる。

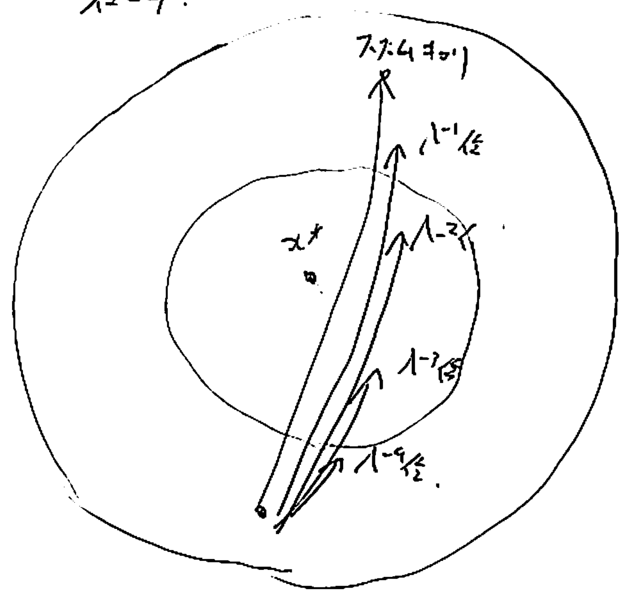
M=2 のとき 2分法と同じ。

（プリントの式から m倍すれば 2次収束になる）

Newton法の
ニガテなもの。



減速ニュートン法のイメージ。



方向はあっているはずなので、
ちぢめていく。

減速ニュートン法。

① できれば普通の Newton法で一到
（2次収束となる）

② 調子が悪ければ、更新幅を短くしていく。

③ $\| f(x^{(k+1)}) \| < \| f(x^{(k)}) \|$

Newton法は減速すべし！

$a_n = a + c_1\lambda_1^n + c_2\lambda_2^n + \cdots$

$1 > |\lambda_1| > |\lambda_2| \cdots$

であって，λ_1 が既知の場合 （c_1 は可知）

a_{n+1} と a_n から $c_1\lambda^n$ を消去する

$$a_{n+1} = a + c_1\lambda_1^{n+1} + c_2\lambda_2^{n+1} + \cdots$$
$$-) \quad \lambda_1 a_n = \lambda_1 a + c_1\lambda_1^{n+1} + c_2\lambda_1\lambda_2^{n+1} \cdots$$
$$a_{n+1} - \lambda_1 a_n = (1-\lambda_1)a + c_2(\lambda_2 - \lambda_1)\lambda_2^n$$

$$\therefore \underbrace{\frac{a_{n+1} - \lambda_1 a_n}{1-\lambda_1}}_{:= a_n^{(1)}} = a + \underbrace{c_2 \frac{\lambda_2 - \lambda_1}{1-\lambda_1}}_{c_2'}\lambda_2^n \cdots$$

あらたな列 $a_n^{(1)}$ $(a_1^{(1)}, a_2^{(1)}, \cdots)$

は a に収束，a_n より速い．

もし，λ_2 も既知なら

$$a_n^{(2)} = \frac{a_{n+1}^{(1)} - \lambda_2 a_n^{(1)}}{1-\lambda_2}$$

Aitken の Δ².

λ_1 が未知．

a_{n+2}, a_{n+1}, a_n から c_1 と λ_1 を求める

$$\lambda_1 \fallingdotseq \frac{a_n - a_{n-1}}{a_{n-1} - a_{n-2}}$$

これを Richardson の Δ² に代入．

$$a_n^{(1)} = a_n - \frac{(a_n - a_{n-1})^2}{a_n - 2a_{n-1} + a_{n-2}}$$