電 301 数値解析
第 3 回
代数方程式

代数方程式 (1)

- 代数方程式は非線形方程式の一種であるが…
- 非線形方程式 $f(x) = 0$ において $f(x)$ を x の多項式に限定したものをを代数方程式という.
- 以下, 変数が複素数である「雰囲気」というを出すために, 変数を z に変える.

代数方程式 (2)

- 複素変数 z の多項式 $f(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_{n-1} z + a_n$ に対して …
- $f(z) = 0$ の複素解をひとつ (あるいはすべて) 求めることを, 代数方程式を解くという.
- 最高次の係数を a_0 とする流儀と, 最高次の係数を a_n とする流儀がある.

代数方程式 (3)

- 回路, 制御, 信号処理などの分野で, 伝達関数で表現されたシステムを取り扱う
- 伝達関数は $\frac{B(s)}{A(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_0}{s^n + a_1 s^{n-1} \cdots + a_n}$ という形
- システムの安定性を調べるためには, 分母多項式の根 ($s^n + a_1 s^{n-1} \cdots + a_n = 0$ の解) を複素数の範囲ですべて求めればよい.

代数方程式 (4)

- 分母多項式の係数から, 根が存在する範囲を大まかに見積ることができる.
- n 次多項式は重複度を含めて n 個の根を持つことがわかっている.
- 「初期値が悪かったから根が求まらなかった」では困る.

代数方程式 (5)

- Scilab では, `roots()` という組み込み関数を使うと代数方程式の解をすべて求めることができる (教科書 48 ページ).

- Scilab には `factors()` という実多項式を 1 次および 2 次の多項式の積に分解する関数もある. 重要なので名前を覚えておくこと.

Scilab のオンラインマニュアル

- これから画面に Scilab のオンラインマニュアルを出して見方を説明する.
- こまめにオンラインマニュアルを見る習慣をつけるとよい.

ソフトウェアの比較 (1)

- 教科書は 2 種類のアルゴリズムを記載
- 第一のアルゴリズムは $\boldsymbol{x}$ が 2 次のベクトルの場合の Newton 法を修正なしで適用したもの.
- 第二のアルゴリズムは Bairstow によるもの (1920 年)

ソフトウェアの比較 (2)

- Newton 法はそのままでは実用にならないが, 考え方を理解するためには重要

- Bairstow 法はすべての根を求められることが特徴だが, 数値的な性質は良くない. 桁落ちなどの影響を受けやすく, 次数が高い多項式では異常終了することがある.

ソフトウェアの比較 (3)

- scilab で多項式の根をすべて求める関数は roots() Jenkins & Traub のアルゴリズム (1970 年) を使用 (この講義では名前の紹介のみ).

- アルゴリズムの詳細に入る前に, Bairstow 法と, Scilab の `roots()`, MATLAB の `roots()`, Mathematica の `NSolve[]` の比較結果を示す.

ソフトウェアの比較 (4)

- 動作環境等は次の通り (どのソフトも 2015.10.19 時点における最新版).
- 最高次の係数が 1 で, 残りの係数が $[-1, 1]$ に値を取る一様分布の擬似乱数から生成された多項式 1000 個の根を求め, 比較した.

ソフトウェアの比較 (5)

- Scilab 5.5.2 for Windows 64bit および MATLAB R2015b for Windows 64bit : Intel Core i5-4690 3.50GHz, 32GB of Memory, Windows7 Professional 64bit Service Pack 1

- Mathematica: Mathematica 10.1.0 for Microsoft Windows (64-bit), Intel(R) Core(TM) i5-2500K CPU 3.30GHz, 16.0GB of Memory, Windows7 Professional 64Bit Service Pack 1

ソフトウェアの比較 (6)

- まずScilabから, Bairstow法とroots()の比較
- Bairstow法は多項式の次数が10のときは動作したが, 20のときは動作しなった (零による除算が発生した可能性がある)
- Scilabは次数100まで一応動作したが …

ソフトウェアの比較 (7)

次数 10 の多項式

	Bairstow 法	roots()
平均求解時間	5.68×10^{-3} 秒	1.25×10^{-4} 秒
$\lvert f(x) \rvert$ の平均	2.16×10^{-10}	9.51×10^{-15}

ソフトウェアの比較 (8)

次数 20 以上の多項式

- 次数 20 にした場合, Bairstow 法は動作しなかった (零による除算?).
- roots() では, 次数 100 まで動作することを確認したが, 次数が上がると間違った解を出力する例が出た.

ソフトウェアの比較 (9)

- $|f(x)| \geq 10^{-3}$ となる x を誤答, それ以外を正解と見做し, 次数を変えたときの, 平均求解時間, 正解に関する $|f(x)|$ の平均, 誤答率, $|f(x)|$ の最悪値を次に示す.

ソフトウェアの比較 (10)

次数 20,50,100 の場合の roots() の結果は …

roots()	次数 20	次数 50	次数 100
平均求解時間	4.68×10^{-5}	5.93×10^{-4}	5.97×10^{-3}
正解の $\lvert f(x)\rvert$ の平均	2.24×10^{-13}	1.28×10^{-7}	5.41×10^{-7}
誤答率	0%	0.004%	0.316%
$\lvert f(x)\rvert$ の最悪値	9.14×10^{-10}	2.05×10^{-2}	8.09×10^{10}

ソフトウェアの比較 (11)

$f(x) = 0$を解いた筈なのに, $f(x)$の絶対値が

$$8.09 \times 10^{10}$$

ソフトウェアの比較 (12)

- 100 次の多項式には根が 100 個
- 100 次の場合の誤答率は 0.3% だから, 平均的には 3 回に 1 回程度おかしな解が混入することになる.

ソフトウェアの比較 (13)

- 「無料のソフトはやっぱり駄目」という話になるかというと …

- MATLAB(295000 円 (教育機関 75000 円, 学生版 4990 円)) および Mathematica(424000 円 (教育機関 192000 円)) と比較

ソフトウェアの比較 (14)

- 100 次の多項式の根を求める (最高次の係数は 1, 残りの係数は区間 $[-1, 1]$ に分布する一様乱数から生成)

- MATLAB では `roots()`,
  Mathematica では `NSolve[]` を使用

ソフトウェアの比較 (15)

	Scilab	MATLAB	Mathematica
平均求解時間	5.97×10^{-3} 秒	4.4×10^{-3} 秒	6.96×10^{-3} 秒
正解の $\lvert f(x)\rvert$ の平均	5.41×10^{-7}	5.67×10^{-7}	3.82×10^{-7}
誤答率	0.316%	0.288%	0.197%
$\lvert f(x)\rvert$ の最悪値	8.09×10^{10}	1.20×10^{12}	1.20×10^{9}

ソフトウェアの比較 (16)

- どのソフトでも解が異常な可能性あり
- 数値計算結果の妥当性をチェックする習慣をつけることが必要. 盲信は危険.
- 教科書等のアルゴリズムを盲信するのも危険 (Bairstow 法の結果を思い出すこと)

ソフトウェアの比較 (17)

なぜこんなことが起こるのか?(1)

- この問題では, 倍精度浮動小数点数では桁数が足りない.
- 一定の条件のもとで収束性が保証されるアルゴリズムでも, そのアルゴリズムが前提条件を満たさない状況で使われることがある.

ソフトウェアの比較 (18)

なぜこんなことが起こるのか?(2)

- 数学的に厳密に収束性が証明できることと一定の数値計算の誤差のもとで解が正解の近傍に収束することは異なる.
- 数値計算の誤差を見込んだ上で解の精度を保証する考え方 (精度保証付き数値計算) もあるが, 必ずしも普及していない.

ソフトウェアの比較 (19)

- 計算手順が定められているのみで, 理論的には収束性が保証されていない「アルゴリズム」もあるので注意

ソフトウェアの比較 (20)

- 実は Mathematica には, 精度保証とは違うが, 力技の対策がある. 計算時間が増えるが, 演算に使う桁数を明示的に指定することで, 精度を高めることができる.

- Mathematica の擬似乱数生成器および `NSolve[]` において `WorkingPrecision->100` として内部計算における桁を 100 桁確保した場合を, 倍精度の場合と比較する.

ソフトウェアの比較 (21)

	Mathematica 倍精度	Mathematica 内部計算 100 桁
平均求解時間	6.96×10^{-3} 秒	1.32×10^{-1} 秒
正解の $\lvert f(x) \rvert$ の平均	3.82×10^{-7}	0.0×10^{-75}
誤答率	0.197%	0%
$\lvert f(x) \rvert$ の最悪値	1.20×10^{9}	0

ソフトウェアの比較 まとめ (1)

- コンピュータによる数値計算の結果は必ずしも信用できないので結果の検証が必要.
- Mathematica では内部計算の桁数を明示的に大きく指定することで浮動小数点数の演算に関する精度不足の問題を解消できる (Maple にもある).

ソフトウェアの比較 まとめ (2)

- 精度保証付き数値計算が使える場合には, ある程度安心.

代数方程式の数値解法 (1)

- Scilab では `roots()` か `factors()` を使う.
- `fsolve()` は実変数のみが対象なので, そのままでは使えない.
- 実用的な数値解法は複雑で初学者向きでないので, この講義では初歩の解説にとどめる.

代数方程式の数値解法 (2)

- もっとも基本的なのは Newton 法.
- 教科書ではなぜか 2 変数版の Newton 法が利用されているが, 複素変数を使えば 1 変数の Newton 法で十分.
- サンプルプログラムを次のシートに示す.

代数方程式の数値解法 (3)

```
deff('[y]=f(x)','y=x^2+1');
deff('[y]=df(x)','y=2*x');
x0=0.9*%i; //初期値, %i は虚数単位
x=x0;maxItr=100;smallTh=1e-8;
for itr=1:maxItr
  x=x-f(x)/df(x);
  if(abs(f(x))<smallTh)
   break();
  end
end
```

代数方程式の数値解法 (4)

- 主要部は `x=x-f(x)/df(x)`
- 1 変数と何も変えなくてよい.
- 初期値を複素数にする.
- `f` と `df` は解きたい方程式とその微分. 関数名は何でもよい.

代数方程式の数値解法 (5)

- 教科書 56~58 ページのプログラムに対応するものは, 次のページのようになる.
- Bairstow 法の説明は略.
- 多項式の因数分解をする場合には, 僅かな数値計算の誤差が大きな問題となることもある.

```
deff('[y]=f(x)','y=x^4-29*x^2-90*x-350');
deff('[y]=df(x)','y=4*x^3-58*x-90');
x0list=[-8,10,-2+2*%i,-2-2*%i];
x=x0list;
maxItr=100;smallTh=1e-8;
for i=1:length(x)
  for itr=1:maxItr
    x(i)=x(i)-f(x(i))/df(x(i));
    if(abs(f(x(i)))<smallTh)
     break();
    end
  end
end
```

代数方程式の数値解法 (6)

- 素朴な Newton 法は初期値次第で発散することがある.
- 直線探索あるいは信頼領域法は「一定の条件のもとで」大域的収束性を保証するが, 重根などの場合に, この「一定の条件」が満たされないこともある.

代数方程式の数値解法 (7)

- Scilab で採用されている Jenkins & Traub のアルゴリズムはポピュラー.
- 連立法 (Durand–Kerner 法および Ehrlich–Aberth 法) と呼ばれる手法もある.
- いずれも複雑なのでこの講義では内容には立ち入らない.

代数方程式の数値解法 (8)

- Newton 法の変形で, 必ずいずれかの解に収束することが保証されたアルゴリズムに, 平野法と呼ばれる方法がある (平野菅保が 1967 年に提案). 以下でこれについて解説する.

平野法 (1)

平野法の特徴

- Taylor 展開の高次項をすべて利用
- 重根にも適用できる
- 初期値をどのように取ってもいずれかの解に大域的に収束する

平野法 (2)

- $p(z) = a_0 z^n + a_1 z^{n-1} + \cdots + z_n$ とし, $p(z) = 0$ を解きたいものとする.

- 仮の解を z_0 とし, $p(z_0 + \zeta)$ を z_0 のまわりで Taylor 展開すると

$$p(z_0 + \zeta) = p(z_0) + \sum_{k=1}^{n} \frac{p^{(k)}(z_0)}{k!} \zeta^k$$

平野法 (3)

$p(z_0+\zeta) = p(z_0) + \sum_{k=1}^{n} \frac{p^{(k)}(z_0)}{k!}\zeta^k$ という展開を利用して …

- $k = 1, 2, \ldots, n$ に対し, Taylor 展開の第 k 次の項 $\frac{p^{(k)}(z_0)}{k!}\zeta^k$ の $p(z_0+\zeta)$ への影響のみを考え, 他の項を無視する. すなわち, 次のように近似する.

$$p(z_0+\zeta) \simeq p(z_0) + \frac{p^{(k)}(z_0)}{k!}\zeta^k$$

平野法 (4)

- $p(z_0+\zeta)=0$ を解きたいので, $0=p(z_0)+\frac{p^{(k)}(z_0)}{k!}\zeta^k$ の解, すなわち

$$\zeta(k)=\left(-\frac{k!p(z_0)}{p^{(k)}(z_0)}\right)^{1/k}$$

を k 番目の解の候補とする (k 乗根は複素数の範囲).

平野法 (5)

- $\zeta(1), \zeta(2), \ldots, \zeta(n)$ を上記の手順で求める.
- $\zeta(1), \zeta(2), \ldots, \zeta(n)$ の中で絶対値がもっとも小さいものが「解の改善の効率が高い」と考え, これを使って z_0 を更新する: $z_0 = z_0 + \zeta(k)$ (ただし $\zeta(k)$ は $\zeta(1), \zeta(2), \ldots, \zeta(n)$ の中で絶対値が最小のもの.

平野法 (6)

- より詳しく言うと, backtracking と呼ばれる手法 (減速とも呼ばれる) を使って, $|p(z_0 + \mu\zeta(k))|$ が $|p(z_0)|$ より小さくなるよう, $\zeta(k)$ に乗ずる適切な「倍率」μ を定める必要がある.

- $\zeta(k)$ は複数の分枝を持つが, その中でもっとも $p(z_0 + \zeta)$ が零に近付くものを採用する.

平野法 (7)

- 平野法は, 以上を, $p(z_0)$ の値が十分小さくなるまで繰り返す解法.

- 代数方程式 $p(z) = 0$ を解くためのアルゴリズムを詳しく書き下すと, 次のようになる.

平野法 (8)

初期化: z の初期値とパラメータ β, λ を定める (ただし $0 < \beta < 1, \lambda > 1$).

ループ: $p(z)$ の絶対値が指定した値以下になるまで以下を繰り返す.

1. μ=1 とする.
2. $k-1, \ldots, n$ に対し, $\zeta(k) = \left(-\mu \frac{k! p(z_0)}{p^{(k)}(z_0)}\right)^{1/k}$ とする.
3. $\{\zeta(1), \ldots, \zeta(n)\}$ の中で絶対値が最小のものを $\zeta(s)$ とする (複数の分枝を含むことがある).
4. $\zeta(s)$ の分枝を $\{\xi_1, \ldots, \xi_s\}$ とする. $\{|p(z+\xi_1)|, \ldots, |p(z+\xi_s)|\}$ の中で最小のものを $|p(z+\xi_t)|$ とする.
5. $|p(z+\xi_t)| < (1-(1-\beta)\mu)p(z)$ なら $z = z+\xi_t$ としてループ冒頭に戻る. そうでなければ $\mu = \mu/\lambda$ としてステップ 2 に戻る.

平野法 (9)

- 収束性については杉原, 室田, 数値計算法の数理, 岩波書店, 1994 などを参照.
- 平野法は, 解をひとつ求めたいときには便利.
- 高次代数方程式の根を多項式の剰余算によって逐次的すべて求めようとすると誤差の集積の問題が発生しやすいので, そのような場合には Jenkins & Traub のアルゴリズムや連立法を使うべき.

Horner の方法と組立除法 (1)

- 次数が高い多項式の値を求めるには, 多数回の乗算と加算が必要になる.
- 高速に多項式の値を評価するためには, 乗算回数を減らすことが望ましい.
- このための方法が Horner の方法 (および組立除法).

Horner の方法と組立除法 (2)

以下の記述の典拠は杉原, 室田, 数値計算法の数理, 岩波書店, 1994.

- 例として, $p(z) = a_0 z^3 + a_1 z^2 + a_2 z + a_3$ において, z に定数 c を代入した値を計算することを考える.
- 次のページのような計算をしてみる. 次ページ 2 行目が Horner の方法.

Horner の方法と組立除法 (3)

- 左の欄の数字に c を掛けたものと上の欄に 1 を掛けたものを足す (右上がりの対角線まで)

	$a_1^0 = a_1$	$a_2^0 = a_2$	$a_3^0 = a_3$
$a_0^1 = a_0$	$a_1^1 = a_0^1 c + a_1^0$	$a_2^1 = a_1^1 c + a_2^0$	$a_3^1 = a_2^1 c + a_3^0$
$a_0^2 = a_0$	$a_1^2 = a_0^2 c + a_1^1$	$a_2^2 = a_1^2 c + a_2^1$	
$a_0^3 = a_0$	$a_1^3 = a_0^3 c + a_1^2$		
$a_0^4 = a_0$			

- 計算してみると …

Horner の方法と組立除法 (4)

- 次のようになるが …

	a_1	a_2	a_3
a_0	$a_0c + a_1$	$a_0c^2 + a_1c + a_2$	$a_0c^3 + a_1c^2 + a_2c + a_3$
a_0	$2a_0c + a_1$	$3a_0c^2 + 2a_1c + a_2$	
a_0	$3a_0c + a_1$		
a_0			

- $p(z) = a_0z^3 + a_1z^2 + a_2z + a_3$ と見比べると …

Horner の方法と組立除法 (5)

	a_1	a_2	a_3
a_0	$a_0c + a_1$	$a_0c^2 + a_1c + a_2$	$\boxed{p(c)}$
a_0	$2a_0c + a_1$	$\boxed{p'(c)}$	
a_0	$\boxed{\frac{1}{2!}p''(c)}$		
$\boxed{\frac{1}{3!}p'''(c)}$			

Horner の方法と組立除法 (6)

- 一般の次数の場合も計算法は同じ.
- 以上のようにすると, 多項式とその導関数に定数 c を代入した値を効率良く求めることができる.