情報数学C
Mathematics for Informatics C

第4回 非線形方程式の求根問題
(2分法,ニュートン法,収束性と初期値,DKA法)

情報メディア創成学類
藤澤誠

今回の講義内容

- **今日の問題**
- 求根問題の数値計算での解き方
- 2分法
- ニュートン法
- 収束性と初期値,DKA法

今回の講義で解く問題

$$f(x) = 0$$

を満たす根xを求めよ

今回の講義で解く問題

xについての方程式$f(x) = 0$の解を求める問題

例1) $f(x) = ax + b = 0$

$f(x)$が**線形方程式**なら解は $x = -b/a$ と簡単に求められる!

例2) $f(x) = 2x^5 + 5x^3 + 3x + 1 = 0$

$f(x)$が**非線形方程式**の場合, 解(根)を求めるのは容易ではない ⇒ 求根問題

(2次方程式なら解の公式, 3次や4次でもカルダノやフェラリの公式でも解けるが, 5次以降は直接解法がない)

今回の講義で解く問題

求根は**どんなところで使われる?**

情報	情報可視化においてある量が一定になる曲面を抽出するのに用いられる($f(\boldsymbol{x}) - T = 0$となる点の抽出). CGでのレイトレーシングなどでも用いられる.
地形・地図	高さ場のデータから等高線を求めるなど
全般	$f(x) = x^2 - 2$とすれば$\sqrt{2}$の値が求まるなど, 数学的に定義されているものの実際の値を求めるときによく使われる

これ自体で問題を解くというより, 問題を解く過程で求根が必要とされることが多い

今回の講義内容

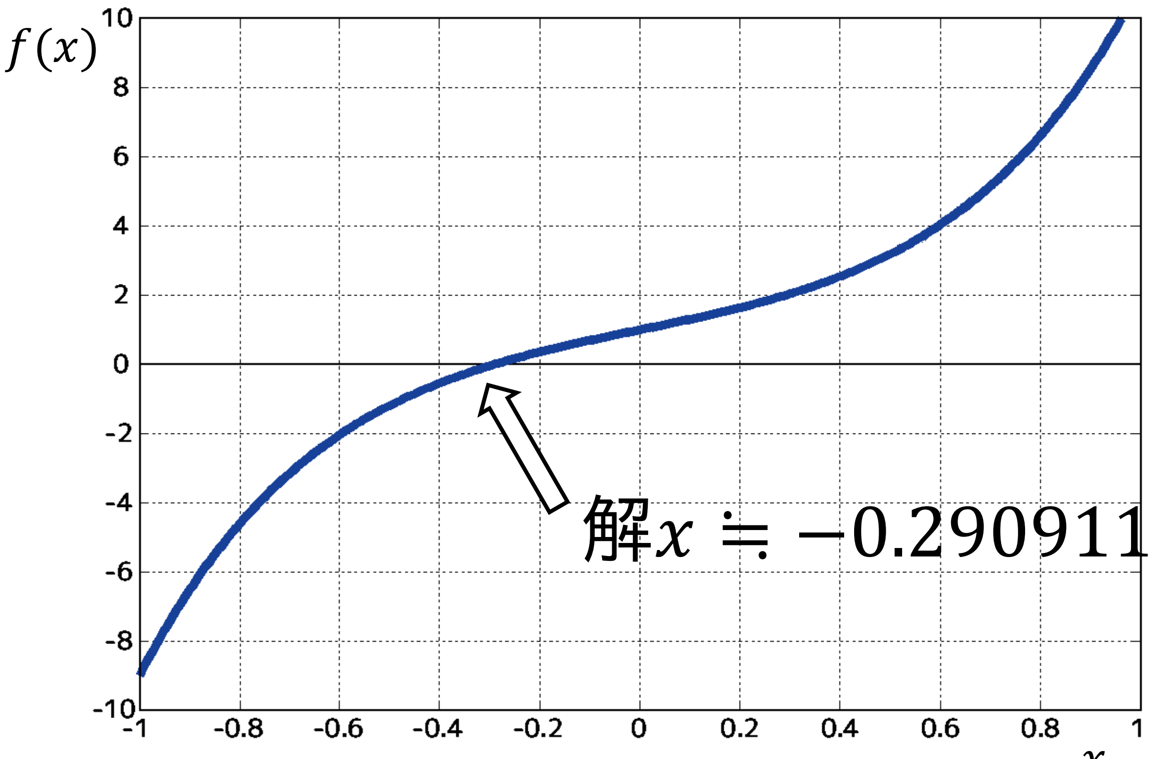
- 今日の問題
- **求根問題の数値計算での解き方**
- 2分法
- ニュートン法
- 収束性と初期値
- ホーナー法, DKA法

求根問題の数値計算での解き方

1回目の講義の復習

$$f(x) = 2x^5 + 5x^3 + 3x + 1 = 0$$

⇒ グラフにすれば解が分かる？

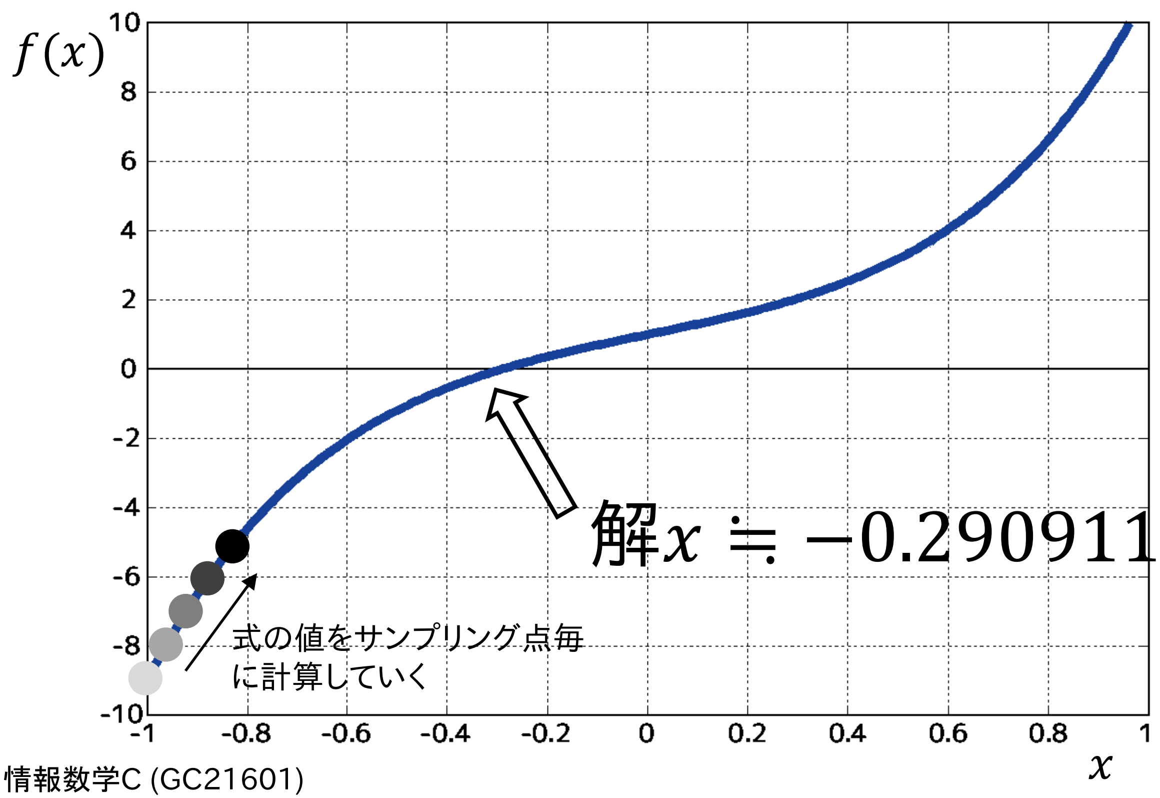


この精度の解をどうやって求めるのか？

求根問題の数値計算での解き方

初期値からサンプリング点を等間隔に取っていく方法だと？

$$f(x) = 2x^5 + 5x^3 + 3x + 1 = 0$$



初期値$x^{(0)} = -1$とする

1×10^{-6}の精度(小数点以下6桁)が必要な場合

$$f(-1) = -9$$
$$f(-0.999999) = -8.99997$$
$$f(-0.999998) = -8.99994$$
$$f(-0.999997) = -8.99992$$
$$\vdots$$

解にたどり着くまでに**70万回以上**$f(x)$を計算しなければならない！

求根問題の数値計算での解き方

等間隔にサンプリング点を取るのは**非効率的**

⇒ どうすれば良いのか？

⇩

適応的にサンプリング間隔を変えれば良い

- 2分法
  ⇒ 関数値のみで計算可能
- ニュートン法
  ⇒ 導関数を使うことで効率的に

今回の講義内容

- 今日の問題
- 求根問題の数値計算での解き方
- **2分法**
- ニュートン法
- 収束性と初期値
- ホーナー法, DKA法

2分法

適応的にサンプリング間隔を変えればより早く解にたどり着く?

⇒ **どのように変えるのか**が重要

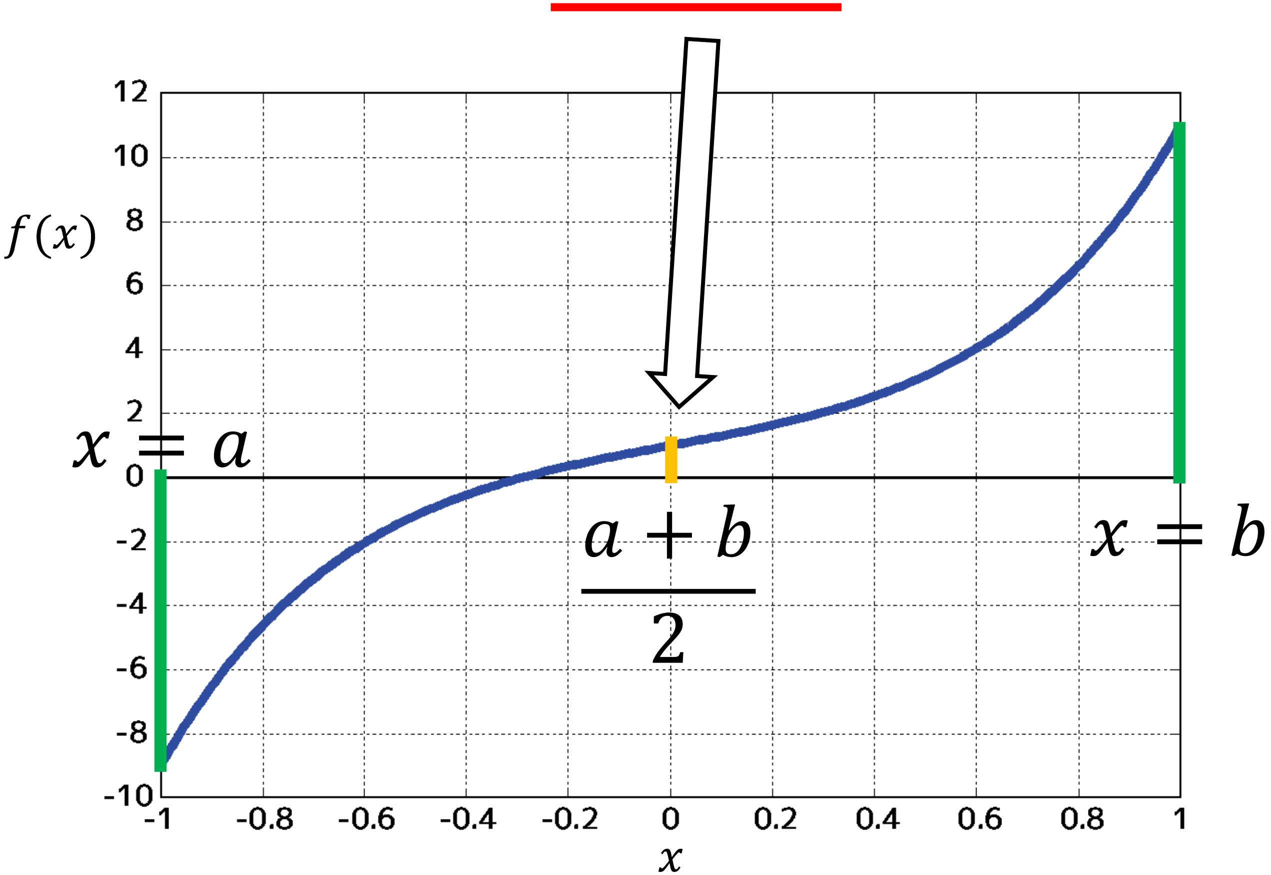
> **前提条件**: $f(a) < 0$および$f(b) > 0$(もしくはこれの逆)であるならば$x = a$**と**$x = b$**の間に**$f(x) = 0$**となる点が少なくとも1つは存在**する(中間値の定理より)

区間$[a, b]$を**2分割**して, それぞれの区間内で解があるかを調べ, 解のある区間をまた2分割する, …としていけば効率がよさそう

2分法

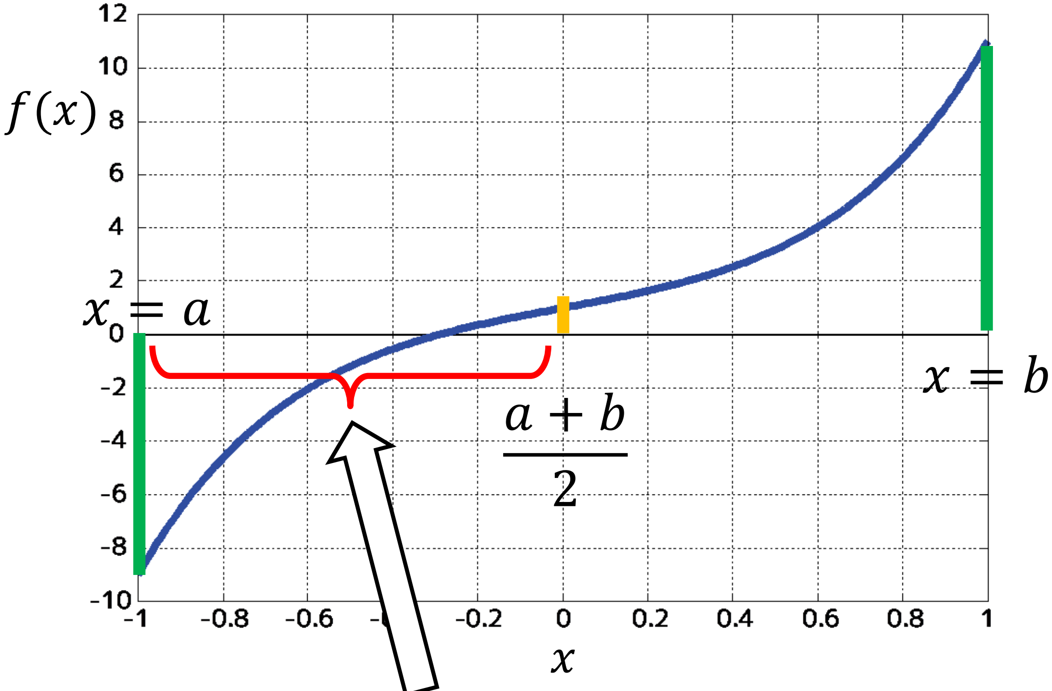
2分法

初期値として$f(a)f(b) < 0$である範囲$[a, b]$が与えられているとき, aとbの中点$\frac{a+b}{2}$を考える



2分法

$f\left(\frac{a+b}{2}\right)$を計算して, その値を$f(a), f(b)$と比べる



$f(a)f\left(\frac{a+b}{2}\right) < 0$の場合:

解は区間$[a, \frac{a+b}{2}]$に存在

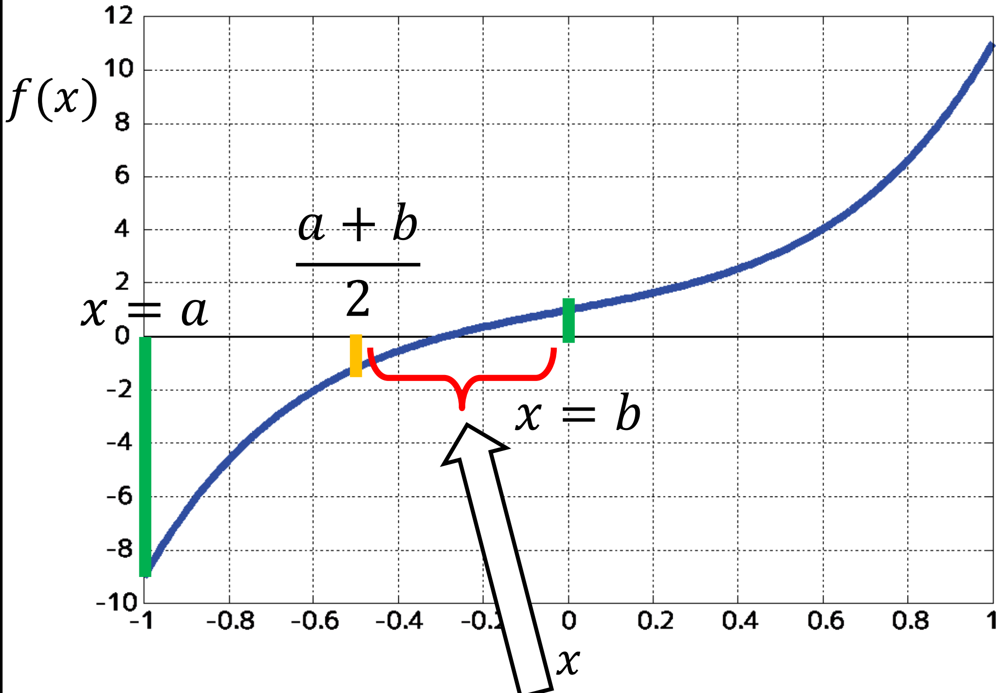
$f\left(\frac{a+b}{2}\right)f(b) < 0$の場合:

解は区間$[\frac{a+b}{2}, b]$に存在

この場合, $f(a)$と$f\left(\frac{a+b}{2}\right)$の符号が異なるので左側の範囲に解があることが分かる

2分法

中点$\frac{a+b}{2}$を新しいa(もしくはb)にして再度中点を計算



新しい中点で同様に以下を判定

$f(a)f\left(\frac{a+b}{2}\right) < 0$の場合：

解は区間$[a, \frac{a+b}{2}]$に存在

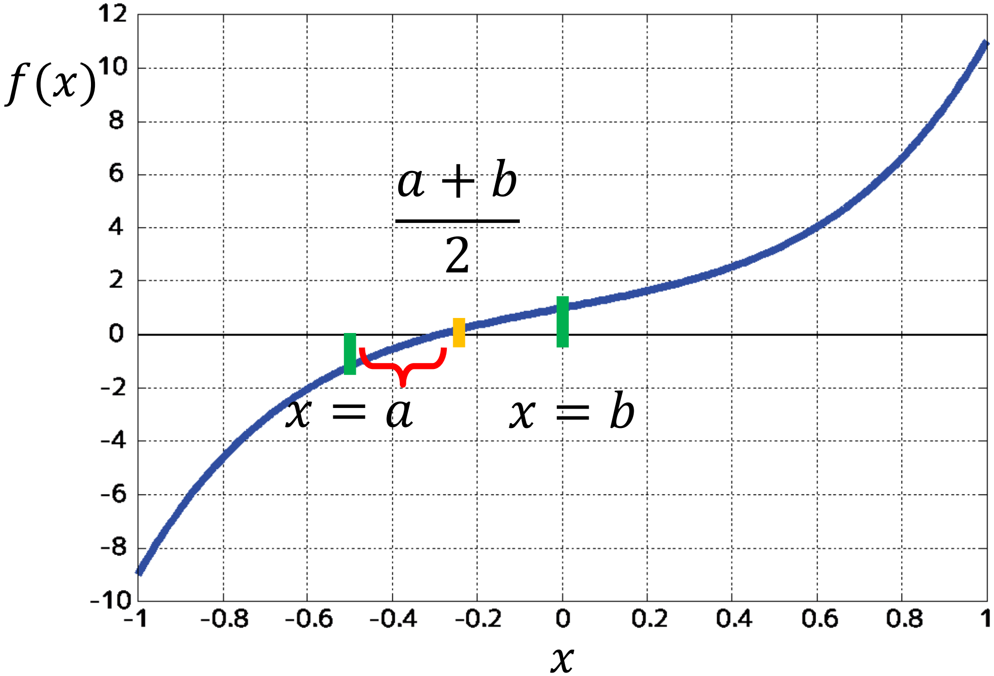
$f\left(\frac{a+b}{2}\right)f(b) < 0$の場合：

解は区間$[\frac{a+b}{2}, b]$に存在

この場合，$f\left(\frac{a+b}{2}\right)$と$f(b)$の符号が異なるので右側の範囲に解があることが分かる

2分法

範囲$[a, b]$の長さが許容誤差以下になるまで処理を繰り返す



区間は毎ステップ1/2になっていくので解の精度を以下の式で予測可能

$$\varepsilon = \frac{b-a}{2^n}$$

注) ここでのa, bは初期区間

⇩ nについて解くと…

許容誤差εになるための繰返し回数: $n = \log_2 \frac{b-a}{\varepsilon}$

(例えば,初期範囲$[-1,1]$で1×10^{-6}の精度を得るには21回反復が必要)

2分法

2分法の計算手順

1. 初期範囲$[a, b]$を設定, $x_l = a, x_r = b$とし, $f(x_l), f(x_r)$を計算しておく
2. 以下の手順を収束するまで繰り返す
   1. 中点$x_{mid} = \frac{x_l + x_r}{2}$における関数値$f(x_{mid})$を計算
   2. $f(x_l)f(x_{mid}) < 0$なら$x_r = x_{mid}$, $f(x_{mid})f(x_r) < 0$なら$x_l = x_{mid}$とする
   3. $|x_r - x_l| < \varepsilon$ or $|f(x_{mid})| < \varepsilon$なら反復終了

中点で$f = 0$となった場合に
対応するための収束条件

関数値$f(x)$の計算は**毎ステップ1回のみ**

2分法

2分法のコード例

```
double f = func(xl);
double fmid = func(xr);
if(f*fmid > 0) return;
double dx = fabs(xr-xl), xmid;
int k;
for(k = 0; k < max_iter; ++k){
    xmid = 0.5*(xl+xr);// 中点
    dx *= 0.5;         // 区間幅を1/2にする
    fmid = func(xmid); // 中点での関数値

    // 収束判定
    if(dx < eps || fmid == 0.0) break;

   // 新しい区間
    if(f*fmid < 0){ xr = xmid;}
    else{ xl = xmid; f = fmid;}
}
x = xmid;
```

初期範囲での$f(x_l), f(x_r)$の計算: 後で値をスワップしないでいいようにf_{mid}に新しい区間での$f(x_r)$を格納 & 初期値チェック

中点とそこでの関数値$f(x_{mid})$の計算:収束判定用の区間幅も計算しておく

収束判定:関数値も収束判定に入れることで$f(x_{mid}) = 0$となった場合に対応

新しい区間の設定:$f \times f_{mid} < 0$で左領域を新しい範囲に($x_r = x_{mid}$), $f \times f_{mid} > 0$で右領域を新しい範囲に($x_l = x_{mid}$). 前者は$f(x_l)$が変わらない, 後者は$f(x_r)$が変わるのでf_{mid}で置き換える必要がある

2分法

2分法の実行結果例

$$f(x) = 2x^5 + 5x^3 + 3x + 1 = 0 \qquad 解\ x \fallingdotseq -0.290911$$

許容誤差$\varepsilon = 1 \times 10^{-6}$,初期探索範囲$[-1,1]$

```
0 : [-1, 1],                   fmid =  1
1 : [-1, 0],                   fmid = -1.1875
2 : [-0.5, 0],                 fmid =  0.169922
3 : [-0.5, -0.25],             fmid = -0.403503
4 : [-0.375, -0.25],           fmid = -0.0960484
5 : [-0.3125, -0.25],          fmid =  0.0414938
6 : [-0.3125, -0.28125],       fmid = -0.0260621
     ⋮ (省略)
16 : [-0.290924, -0.290894], fmid =  1.15822e-05
17 : [-0.290924, -0.290909], fmid = -2.15376e-05
18 : [-0.290916, -0.290909], fmid = -4.97764e-06
19 : [-0.290913, -0.290909], fmid =  3.30229e-06
20 : [-0.290913, -0.290911], fmid = -8.37673e-07
x = -0.290912
```

21回の反復で処理終了

2分法

2分法の実行結果例(2分法によるπの計算)

$$f(x) = \cos\left(\frac{x}{2}\right) = 0 \qquad \text{解 } x = \pi = 3.14159265\cdots$$

許容誤差$\varepsilon = 1 \times 10^{-6}$,初期探索範囲[3,4]

```
0 : [3, 4], fmid = -0.178246
1 : [3, 3.5], fmid = -0.0541771
2 : [3, 3.25], fmid = 0.00829623
3 : [3.125, 3.25], fmid = -0.0229517
4 : [3.125, 3.1875], fmid = -0.00732861
5 : [3.125, 3.15625], fmid = 0.000483827
 ⋮ (省略)
15 : [3.14157, 3.1416], fmid = 3.17494e-06
16 : [3.14159, 3.1416], fmid = -6.39758e-07
17 : [3.14159, 3.14159], fmid = 1.26759e-06
18 : [3.14159, 3.14159], fmid = 3.13916e-07
19 : [3.14159, 3.14159], fmid = -1.62921e-07
x = 3.14159
```

20回の反復で処理終了

*1回目の授業で紹介したライプニッツの公式による円周率πの計算では1万項まで計算して10^{-4}程度の誤差

今回の講義内容

- 今日の問題
- 求根問題の数値計算での解き方
- 2分法
- **ニュートン法**
- 収束性と初期値
- ホーナー法, DKA法

ニュートン法

2分法より収束を早めるには?

- 分割数を増やしてみる?
  例) 初期範囲$[-1,1]$で1×10^{-6}の精度を得るための反復回数は?
  2分法:21回, 3分法:14回, 4分法:11回
  ⇒ 反復回数は減らせそうだけど…

分割数を増やすと毎反復での$f(x)$の**計算回数が増える**(2分法だと1回, 3分法だと2回, …)
⇒ **総計算量が増える**可能性あり

ニュートン法

分割数を増やすのではない別の方法が必要

- 現在の関数値$f(x)$を使えないか？
  $f(x)$の絶対値が大きいときは分割幅を大きく，
  小さいときに分割幅を小さく
  ⇒ 絶対値だと**分割幅の設定が難しい**
- 関数値の**相対変化量**を使えないか？

$$\text{相対変化量：}\frac{f(x+\Delta x)-f(x)}{\Delta x}$$

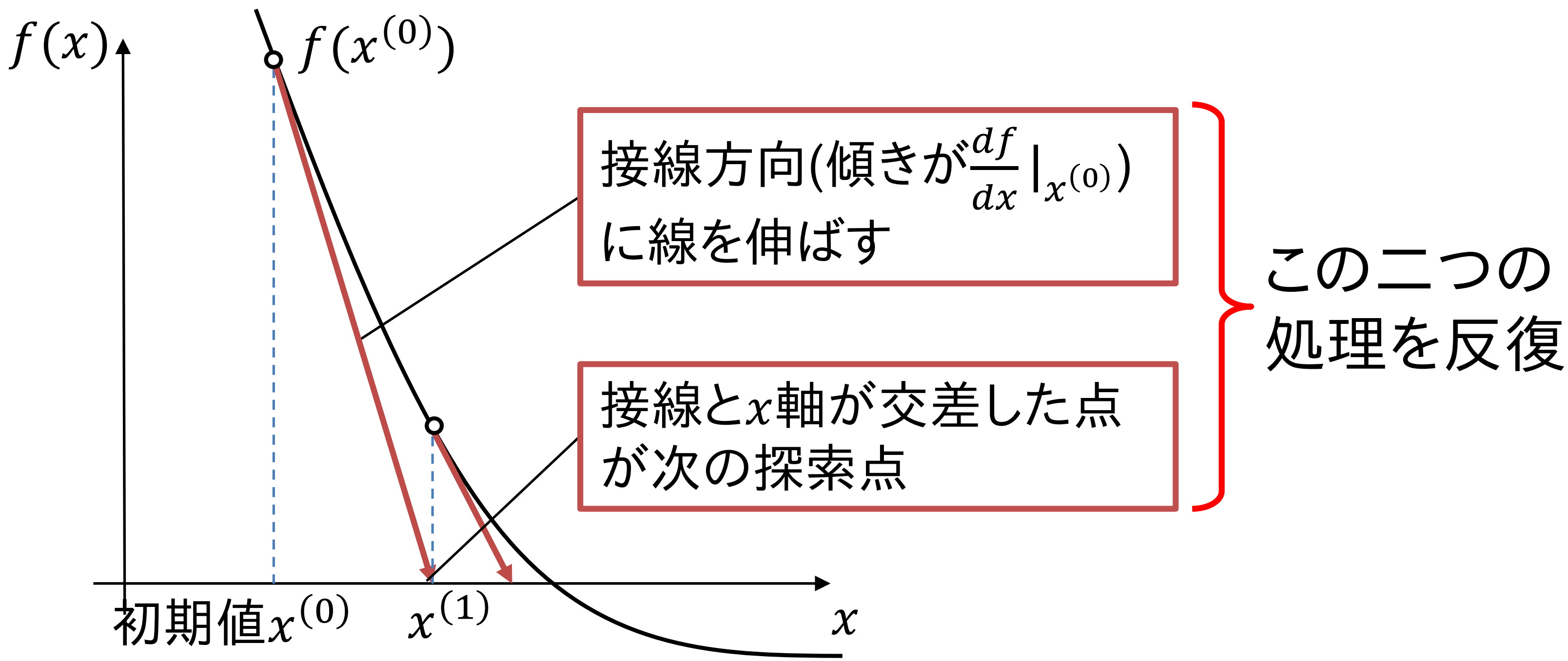
$\Delta x \to 0$の極限をとると $\dfrac{df}{dx}$

ニュートン法

ニュートン法

ニュートン法(ニュートン・ラフソン法)の考え方

- 初期値$x^{(0)}$からスタート (範囲$[a,b]$ではなく初期値のみ)
- 関数の微分値$\frac{df}{dx}$に基づいて次の値$x^{(k+1)}$を決定



ニュートン法

ニュートン法の処理を式にしてみよう

- 接線方向(傾きが$\frac{df}{dx}\big|_{x^{(0)}}$)に線を伸ばす

  $(x^{(0)}, f(x^{(0)}))$での接線：$y = f'\left(x^{(0)}\right)\left(x - x^{(0)}\right) + f\left(x^{(0)}\right)$

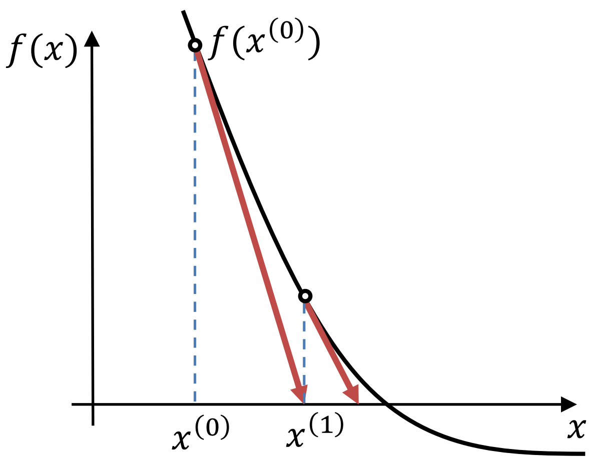
  (ただし, $f'(x) = df/dx$としている)

- 接線とx軸が交差した点が次の探索点
  次の探索点(接線の式で$y = 0$となる点)：

$$x^{(1)} = x^{(0)} - \frac{f\left(x^{(0)}\right)}{f'\left(x^{(0)}\right)}$$

⇩ kステップ目とすると

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



ニュートン法

ニュートン法の計算手順

1. 初期値$x^{(0)}$を設定
2. 以下の手順を収束するまで繰り返す($k = 0 \sim$)
   1. $x^{(k)}$での関数値$f(x^{(k)})$と導関数値(傾き) $f'(x^{(k)})$ を計算
   2. 次の計算点$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}$を求める
   3. $\left|x^{(k+1)} - x^{(k)}\right| < \varepsilon$ もしくは $\left|f(x^{(k+1)})\right| < \varepsilon$ なら反復終了

ニュートン法

ニュートン法のコード例

```
int k;
for(k = 0; k < max_iter; ++k){
    // 現在の位置xにおける関数値と導関数の計算
    f = func(x);
    df = dfunc(x);

    // 導関数の結果から次の位置を計算
    x = x-f/df;

    // 収束判定
    dx = fabs(f/df);
    if(dx < eps || fabs(f) < eps){
        break;
    }
}
```

$x^{(k)}$での関数値と導関数の計算

$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}$で次の計算点位置を求める

安定性をより高めたいならこの処理の前にdfが0でないかの判定を入れる

ニュートン法

ニュートン法の実行結果例

$$f(x) = 2x^5 + 5x^3 + 3x + 1 = 0 \qquad 解\ x \fallingdotseq -0.290911$$

許容誤差$\varepsilon = 1 \times 10^{-6}$,初期値$x^{(0)} = -1$

```
0 : f(-1) = -9
1 : f(-0.678571) = -2.88573
2 : f(-0.438636) = -0.770354
3 : f(-0.315502) = -0.109784
4 : f(-0.291595) = -0.00296924
5 : f(-0.290912) = -2.26919e-06
x = -0.290911
```

6回の反復で処理終了

2分法での反復回数21回から6回へと大幅に減っている

ニュートン法

ニュートン法の特徴

- 2分法と違い**初期値のみ**でOK
- **導関数が必要**
  - 1反復内での計算量が増える?
    ⇒ 導関数は元の関数より次数が減ることが多い
    ⇒ 計算量が増えても**反復回数はそれ以上に減って**いるので問題なし
  - 離散データなどで**導関数を数学的に計算できない**場合は?

セカント法(割線法)

離散的なデータを扱う場合, 必ずしも**導関数が数学的に計算できない**

⇒ 導関数を何らかの形で**近似**する必要がある

⇩

微分の代わりに**差分**を使う

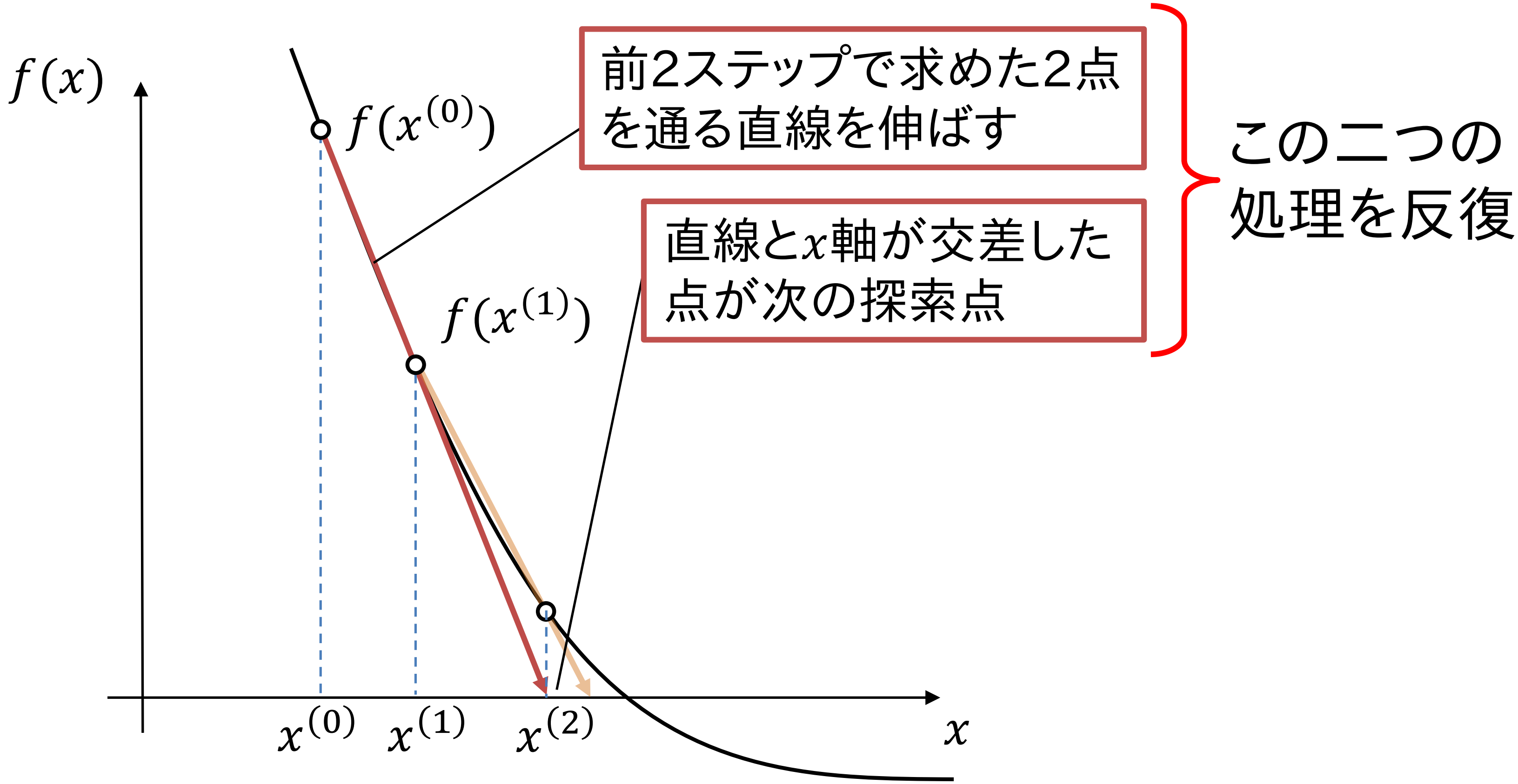
$$f'\left(x^{(k)}\right) \simeq \frac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}$$

xの更新式： $x^{(k+1)} = x^{(k)} - f\left(x^{(k)}\right) \dfrac{x^{(k)} - x^{(k-1)}}{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}$

セカント法(割線法)

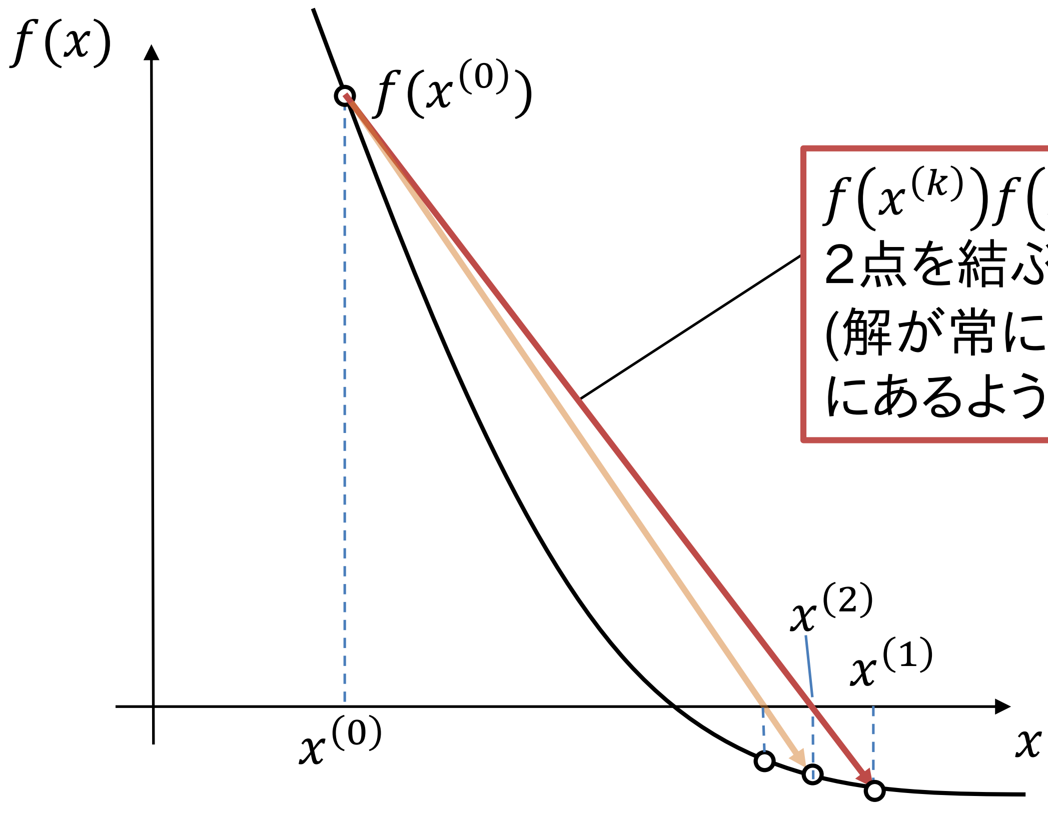
セカント法(割線法)

セカント法(割線法)の考え方



挟み撃ち法

同様の計算方法として**挟み撃ち法**もある



$f\left(x^{(k)}\right)f\left(x^{(k+1)}\right) < 0$となる2点を結ぶ直線を常に使う(解が常に$x^{(k)}$と$x^{(k+1)}$の間にあるようにする)

セカント法(割線法)

セカント法(割線法)の計算手順

1. 初期値$x^{(0)}, x^{(1)}$を設定
2. 以下の手順を収束するまで繰り返す($k = 1 \sim$)
    1. $x^{(k)}$での関数値$f(x^{(k)})$を計算 or $x^{(k)}$での値$f(x^{(k)})$を離散データから取り出す
    2. 次の計算点$x^{(k+1)} = x^{(k)} - f\left(x^{(k)}\right)\frac{x^{(k)}-x^{(k-1)}}{f\left(x^{(k)}\right)-f\left(x^{(k-1)}\right)}$を求める
    3. $\left|x^{(k+1)} - x^{(k)}\right| < \varepsilon$ もしくは $\left|f(x^{(k+1)})\right| < \varepsilon$ なら反復終了

手順自体はニュートン法と同じなのでコード例はなし

ニュートン法

ニュートン法の特徴

- 2分法と違い**初期値のみ**でOK
- **導関数が必要**
  - 1反復内での計算量が増える?
    ⇒ 総計算時間は減る
  - 離散データなどで**導関数を数学的に計算できない場合**は?
    ⇒ セカント法
- **多次元への拡張**は可能か?

多次元のニュートン法

n次元$(x_0, x_1, \dots, x_{n-1})$のニュートン法は？

変数がn個, 式もn個ある場合:

$$\begin{cases} f_0(x_0, x_1, \dots, x_{n-1}) = 0 \\ f_1(x_0, x_1, \dots, x_{n-1}) = 0 \\ \vdots \\ f_{n-1}(x_0, x_1, \dots, x_{n-1}) = 0 \end{cases}$$

⇩ ベクトル表記

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$$

$\boldsymbol{f}, \boldsymbol{x}, \boldsymbol{0}$はベクトル(太字表記)になっていることに注意

ここで $\boldsymbol{x} = (x_0, x_1, \dots, x_{n-1})^T$

$$\boldsymbol{f}(\boldsymbol{x}) = \left(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \dots, f_{n-1}(\boldsymbol{x})\right)^T$$

多次元のニュートン法

kステップ目の近似値$\boldsymbol{x}^{(k)}$の周りでテイラー展開

$$\boldsymbol{f}\left(\boldsymbol{x}^{(k+1)}\right) = \boldsymbol{f}\left(\boldsymbol{x}^{(k)}\right) + \boldsymbol{f}'\left(\boldsymbol{x}^{(k)}\right)\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right) + \cdots$$

$n \geq 2$の項を無視して($\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)$の値が小さければ無視できる),
$\boldsymbol{x}^{(k+1)}$について解くと：

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

この式で解けるのか？ f'って何？

参考) 関数$f(x)$のa周りのテイラー展開：

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \cdots$$

多次元のニュートン法

多変数関数のテイラー展開から$\boldsymbol{f}'(x)$は**ヤコビ行列**J

$$\boldsymbol{f}'(\boldsymbol{x}) = J(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_0}{\partial x_0} & \frac{\partial f_0}{\partial x_1} & \cdots & \frac{\partial f_0}{\partial x_{n-1}} \\ \frac{\partial f_1}{\partial x_0} & \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{n-1}}{\partial x_0} & \frac{\partial f_{n-1}}{\partial x_1} & \cdots & \frac{\partial f_{n-1}}{\partial x_{n-1}} \end{pmatrix}$$

$\boldsymbol{f}'(x)$で割る = $J(x)$の逆行列を掛ける

参考) 2変数関数のテイラー展開：

$$f(x_0, x_1) = \sum_{n=0}^{\infty}\sum_{m=0}^{\infty} \frac{((x_0 - a_0)^n (x_1 - a_1)^n)}{n!\,m!}\left(\frac{\partial^{n+m} f}{\partial x_0^n \partial x_1^m}\right)$$

多次元のニュートン法

・求めたいのは$\boldsymbol{x}^{(k)}$から$\boldsymbol{x}^{(k+1)}$への変化量
　$\Rightarrow \boldsymbol{\delta} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}$とする

・求根問題なので$\boldsymbol{f}\left(\boldsymbol{x}^{(k+1)}\right) = \boldsymbol{0}$になってほしい

テイラー展開した式$\left(\boldsymbol{f}\left(\boldsymbol{x}^{(k+1)}\right) = \boldsymbol{f}\left(\boldsymbol{x}^{(k)}\right) + \boldsymbol{f}'\left(\boldsymbol{x}^{(k)}\right)\left(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\right)\right)$より:

$$J\left(\boldsymbol{x}^{(k)}\right)\boldsymbol{\delta} = -\boldsymbol{f}\left(\boldsymbol{x}^{(k)}\right)$$

$\Rightarrow$ $\boldsymbol{\delta}$を未知変数とした**線形システム**なので**ガウスの消去法や三角分解で解ける**

$\boldsymbol{\delta}$について解いた後に $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{\delta}$ で$\boldsymbol{x}$を更新

多次元のニュートン法

多次元版ニュートン法の計算手順

1. 初期値ベクトル$\boldsymbol{x}^{(0)}$を設定
2. 以下の手順を収束するまで繰り返す$(k = 0 \sim)$
   1. $\boldsymbol{x}^{(k)}$での関数値$\boldsymbol{f}\left(\boldsymbol{x}^{(k)}\right)$とヤコビ行列$J\left(\boldsymbol{x}^{(k)}\right)$の各要素を計算
   2. $J(\boldsymbol{x})\boldsymbol{\delta} = -\boldsymbol{f}(\boldsymbol{x}^{(k)})$をガウスの消去法などで解く
   3. $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{\delta}$ で$\boldsymbol{x}$を更新
   4. $|\boldsymbol{\delta}| < \varepsilon$ もしくは $\left|\boldsymbol{f}(\boldsymbol{x}^{(k+1)})\right| < \varepsilon$ なら反復終了

多次元のニュートン法

多次元のニュートン法のコード例

Jは$n \times (n+1)$の配列, 関数$f_i(\boldsymbol{x})$は構造体を使ってfunc[i].func(x)と表している.

導関数$f_i'(\boldsymbol{x})$はfunc[i].dfunc(x) ⇐ 返値が$\left(\frac{\partial f_i}{\partial x_0}, \dots, \frac{\partial f_i}{\partial x_{n-1}}\right)$のベクトル(配列)

```
int k;
for(k = 0; k < max_iter; ++k){
    // 現在の位置xにおける関数値の計算
    for(int i = 0; i < n; ++i) f[i] = -funcs[i].func(x);
    // ヤコビ行列の計算
    for(int i = 0; i < n; ++i){
        df = funcs[i].dfunc(x);
        for(int j = 0; j < n; ++j) J[i][j] = df[j];
    }
    // 線形システムを解いてδを計算
    for(int i = 0; i < n; ++i) J[i][n] = f[i];
    GaussEliminationWithPivoting(J, n);
    // xを更新
    for(int i = 0; i < n; ++i) x[i] += J[i][n];

    // 収束判定
    d = 0.0;
    for(int i = 0; i < n; ++i) d += fabs(f[i]);
    if(d < eps) break;
}
```

線形システムの係数行列と右辺項ベクトルの計算

ガウスの消去法で線形システムを解く

$\boldsymbol{x}$の更新($\boldsymbol{\delta}$は拡大行列Jのn列目に格納されることに注意)

多次元のニュートン法

多次元のニュートン法の実行結果例

$$\begin{cases} f_0(x,y) = x^2 - 4xy + y^2 = 0 \\ f_1(x,y) = x^2 + y^2 - 2 = 0 \end{cases}$$

$$\left(\frac{\partial f_0}{\partial x} = 2x - 4y, \frac{\partial f_0}{\partial y} = -4x + 2y,\ \frac{\partial f_1}{\partial x} = 2x, \frac{\partial f_1}{\partial y} = 2y \right)$$

許容誤差$\varepsilon = 1 \times 10^{-6}$,初期値$(x^{(0)}, y^{(0)}) = (1, 0)$

```
0 : f0(1,      0)        = -1,          f1(1,      0)        = 1
1 : f0(1.5,    0.5)      = 0.5,         f1(1.5,    0.5)      = -0.5
2 : f0(1.375,  0.375)    = 0.03125,     f1(1.375,  0.375)    = -0.03125
3 : f0(1.36607,0.366071) = 0.000159439, f1(1.36607,0.366071) = -0.000159439
4 : f0(1.36603,0.366025) = 4.23634e-09, f1(1.36603,0.366025) = -4.23634e-09
(x1,x2) = (1.36603, 0.366025)
```

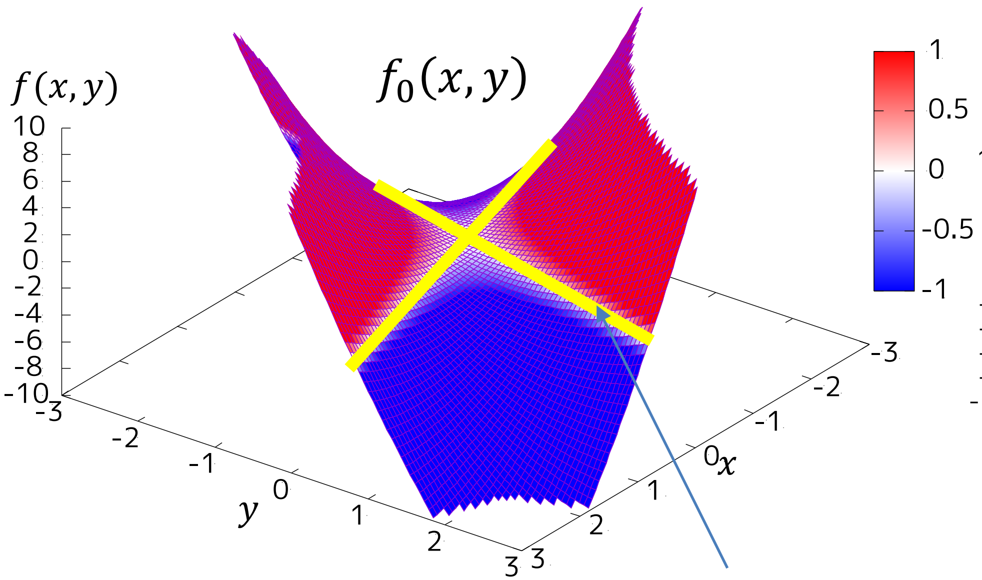
5回の反復で処理終了

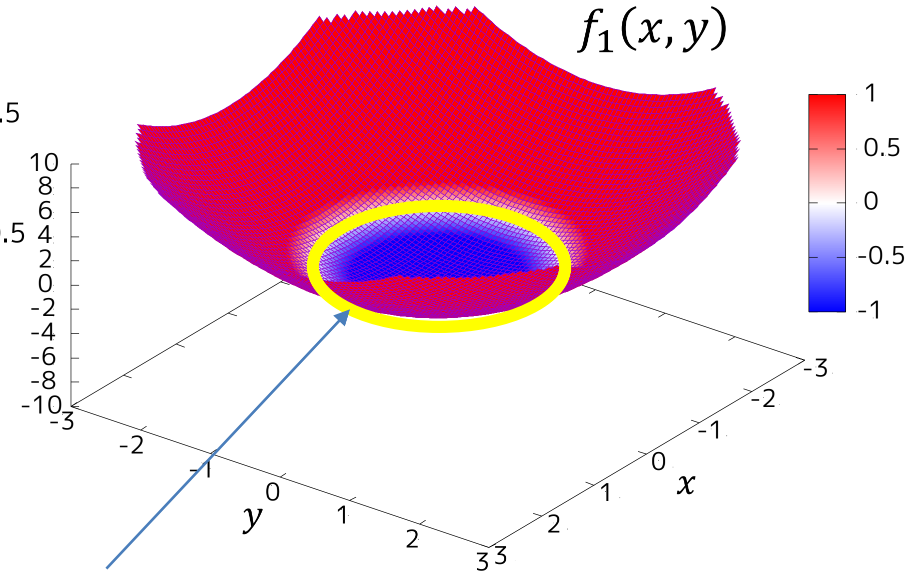
解は一つだけ?

多次元のニュートン法

各関数の形を見てみよう

$$\begin{cases} f_0(x,y) = x^2 - 4xy + y^2 = 0 \\ f_1(x,y) = x^2 + y^2 - 2 = 0 \end{cases}$$





これらが$f(x,y) = 0$の断面

多次元のニュートン法

xy平面を見てみる:

$$\begin{cases} f_0(x,y) = x^2 - 4xy + y^2 = 0 \\ f_1(x,y) = x^2 + y^2 - 2 = 0 \end{cases}$$

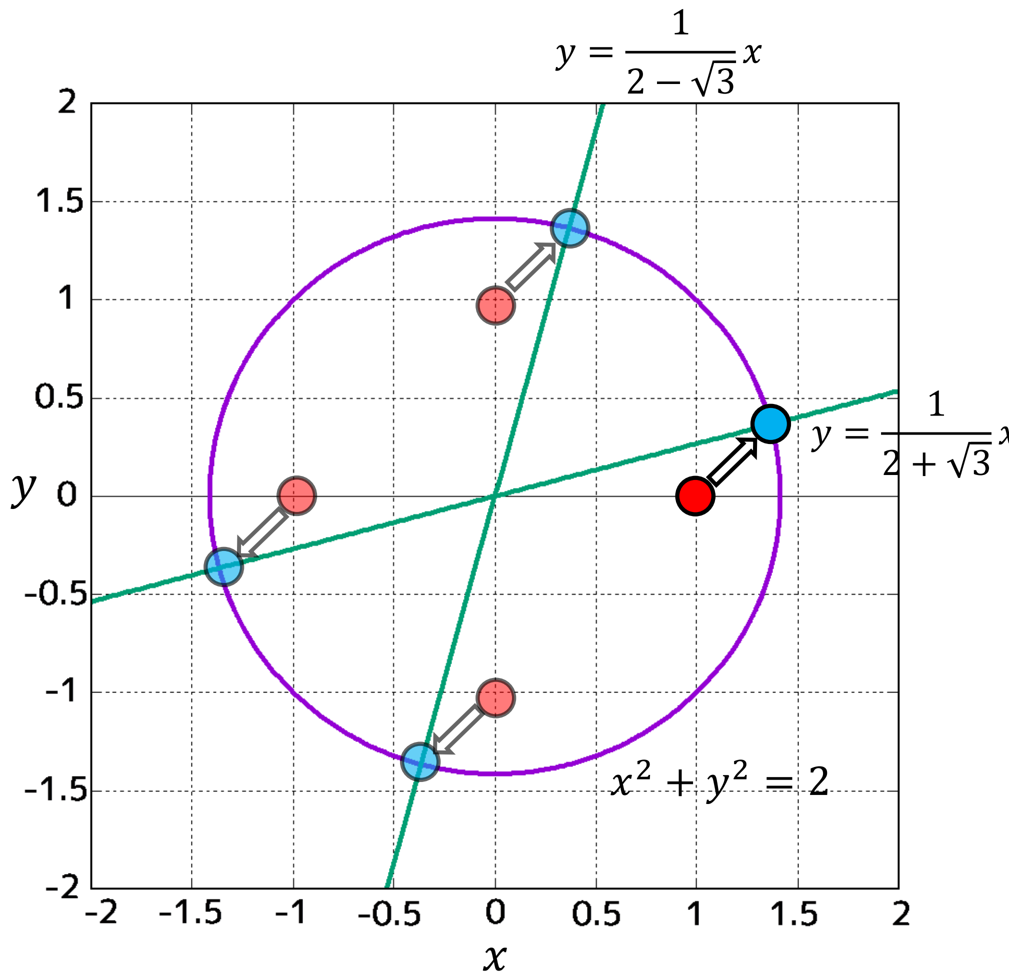
⇩ $f_i(x,y) = 0$の平面上だと

$$\begin{cases} y = \dfrac{1}{2 \pm \sqrt{3}} x \\ x^2 + y^2 = 2 \quad (\text{半径}\sqrt{2}\text{の円}) \end{cases}$$

初期値$(x^{(0)}, y^{(0)}) = (1, 0)$で
解$(x,y) = (1.36603, 0.366025)$
が得られた

⇩

初期値を変えてやれば
他の解も得られそう



多次元のニュートン法

初期値を変えた結果

$$\begin{cases} f_0(x,y) = x^2 - 4xy + y^2 = 0 \\ f_1(x,y) = x^2 + y^2 - 2 = 0 \end{cases}$$

初期値$(x^{(0)}, y^{(0)}) = (0, 1)$

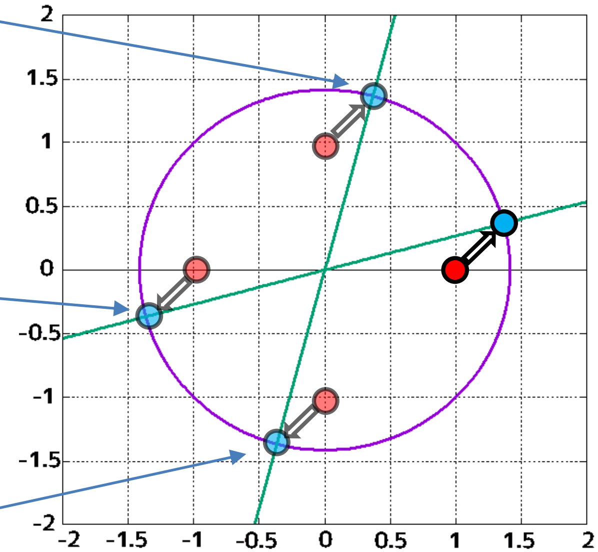
```
0 : f0(0,        1)        = -1,           f1(0,        1)        = 1
1 : f0(0.5,      1.5)      = 0.5,          f1(0.5,      1.5)      = -0.5
2 : f0(0.375,    1.375)    = 0.03125,      f1(0.375,    1.375)    = -0.03125
3 : f0(0.366071,1.36607) = 0.000159439, f1(0.366071,1.36607) = -0.000159439
4 : f0(0.366025,1.36603) = 4.23634e-09, f1(0.366025,1.36603) = -4.23634e-09
(x1,x2) = (0.366025, 1.36603)
```

初期値$(x^{(0)}, y^{(0)}) = (-1, 0)$

```
0 : f0(-1,      0)          = -1,           f1(-1,      0)          = 1
1 : f0(-1.5,    -0.5)       = 0.5,          f1(-1.5,    -0.5)       = -0.5
2 : f0(-1.375,  -0.375)     = 0.03125,      f1(-1.375,  -0.375)     = -0.03125
3 : f0(-1.36607,-0.366071) = 0.000159439, f1(-1.36607,-0.366071) = -0.000159439
4 : f0(-1.36603,-0.366025) = 4.23634e-09, f1(-1.36603,-0.366025) = -4.23634e-09
(x1,x2) = (-1.36603, -0.366025)
```

初期値$(x^{(0)}, y^{(0)}) = (0, -1)$

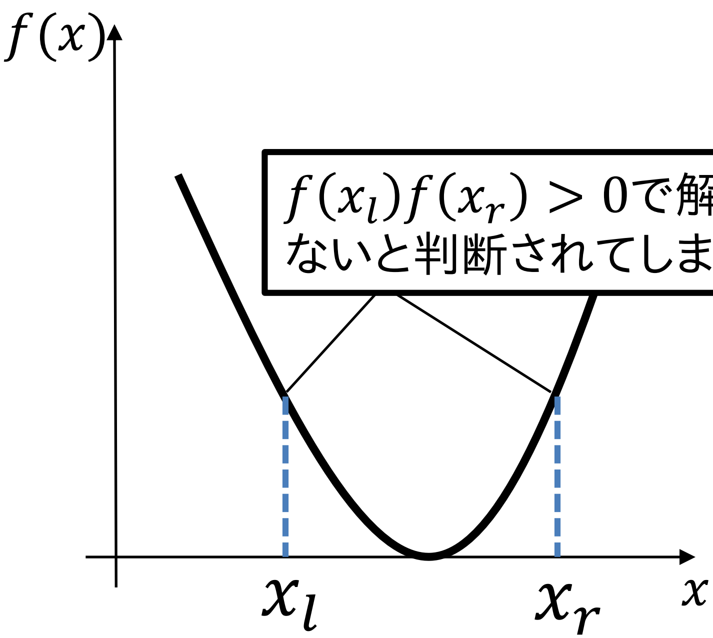
```
0 : f0(0,         -1)        = -1,           f1(0,         -1)        = 1
1 : f0(-0.5,      -1.5)      = 0.5,          f1(-0.5,      -1.5)      = -0.5
2 : f0(-0.375,    -1.375)    = 0.03125,      f1(-0.375,    -1.375)    = -0.03125
3 : f0(-0.366071,-1.36607) = 0.000159439, f1(-0.366071,-1.36607) = -0.000159439
4 : f0(-0.366025,-1.36603) = 4.23634e-09, f1(-0.366025,-1.36603) = -4.23634e-09
(x1,x2) = (-0.366025, -1.36603)
```



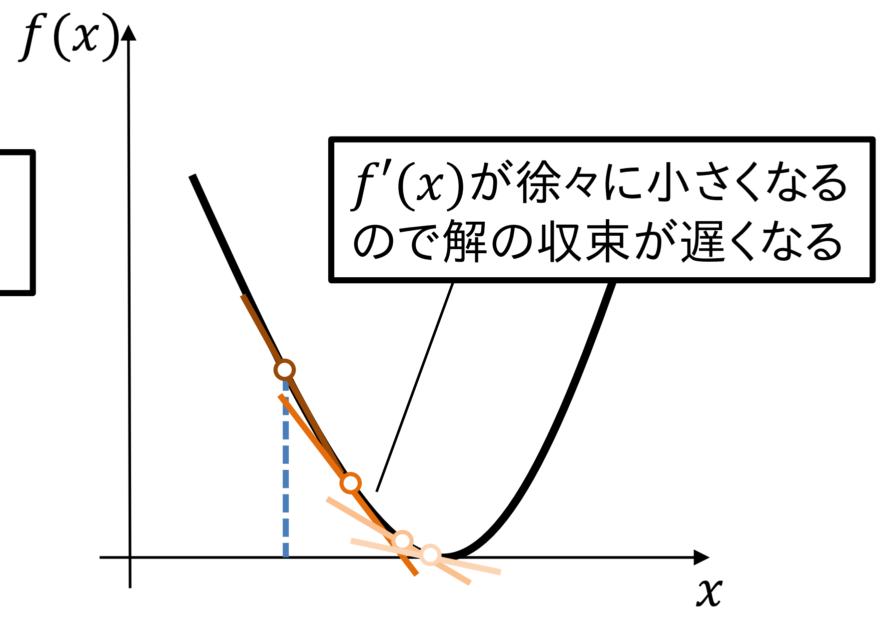
ニュートン法

重根の場合でも解は求まる?

- **2分法**:根の前後の符号が同じになるので✕
- **ニュートン法** : **収束は遅く**なるけど**解は求まる**



2分法の場合



ニュートン法の場合

今回の講義内容

- 今日の問題
- 求根問題の数値計算での解き方
- 2分法
- ニュートン法
- **収束性と初期値**
- ホーナー法, DKA法

収束性と初期値

解に**収束するための条件**は?**どのくらいの早さで収束**する?

⇒ 不動点定理で解への収束条件が分かる

バナッハの不動点定理(縮小写像の定理)

閉区間I内での写像*を行う関数gについて

$$\|g(x_1) - g(x_2)\| \ \leq \ L\|x_1 - x_2\|$$

が成り立つようなL $(0 \leq L < 1)$が存在するとき,

$g(x)$は**縮小写像**という(**リプシッツ条件**).

このときのLをリプシッツ定数という.

(定理は次のページに続く)

*写像は2つの集合間の関係を表すもので, 例えば関数$y = f(x)$も$x \in \Re$から$y \in \Re$への写像$(\Re \to \Re)$といえる.

収束性と初期値

バナッハの不動点定理のつづき

$g(x)$が縮小写像なら, $x^* = g(x^*)$となる**不動点を必ず持つ**.

(定理はここまで)

この写像が例えばニュートン法の更新式だったとすると:

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

もし, $g(x)$がリプシッツ条件を満たすならば, K回目の反復で $x^{(K)} = g(x^{(K)})$となる不動点が存在

$\Rightarrow$ 要するに$L < 1$ならいつか解に収束するということ

*不動点定理の証明は[Wikipedia]など参照

収束性と初期値

リプシッツ条件を扱いやすいように**シンプル**にしてみる

閉区間*内の2点x_1, x_2について$g(x)$がその区間内で微分可能ならば, 平均値の定理より,

$$g(x_1) - g(x_2) = g'(\xi)(x_1 - x_2)$$

となるξがx_1とx_2の間に存在する.
リプシッツ条件$\|g(x_1) - g(x_2)\| \le L\|x_1 - x_2\|$から,

$$\|g(x_1) - g(x_2)\| = \|g'(\xi)\|\|x_1 - x_2\| \le L\|x_1 - x_2\|$$

⇩

$$\|g'(x)\| \le L$$

を満たすならば, $g(x)$はリプシップ条件を満たす
($0 \le L < 1$なので$\|g'(x)\| < 1$が条件になる)

*閉区間は1次元なら$[a, b]$のような範囲と考えればよい

収束性と初期値

ニュートン法における収束条件

$x^{(k)}$をxとすると更新式は： $g(x) = x - \frac{f(x)}{f'(x)}$

gをxで微分すると

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

$$\|g'(x)\| = \left\| f(x) \frac{f''(x)}{\left(f'(x)\right)^2} \right\| < 1$$ ：ニュートン法の収束条件

注) あくまでこの条件は**局所収束性**で**大域収束性**ではない

例) $f(x) = ax^2$の場合, $g'(x) = ax^2 \frac{(2a)}{(2ax)^2} = \frac{1}{2} < 1$なので安定

⇨ ax^2は凸関数の一種(この辺の話は次回に)

収束性と初期値

2分法, ニュートン法の**収束速度は?**

真値をαとするとkステップ目の誤差は:

$$\epsilon^{(k)} = x^{(k)} - \alpha$$

2分法の誤差収束

1ステップ毎に範囲が$1/2$になるので,

$$\epsilon^{(k+1)} = \frac{\epsilon^{(k)}}{2}$$

誤差が1反復処理毎に定数倍(< 1)となっているので,
2分法は**1次収束**

収束性と初期値

ニュートン法の誤差収束(単根の場合)

$\epsilon^{(k)} = x^{(k)} - \alpha$をニュートン法の更新式$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$に代入

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

$f(x^{(k)})$を真値α周りでテイラー展開($x^{(k)}$がαに近いと仮定)

$$f\left(x^{(k)}\right) \simeq \underbrace{f(\alpha)}_{f(\alpha)=0} + f'(\alpha)\underbrace{\left(x^{(k)} - \alpha\right)}_{\epsilon^{(k)}} + \frac{f''(\alpha)\left(x^{(k)} - \alpha\right)^2}{2!}$$

$$f\left(x^{(k)}\right) \simeq f'(\alpha)\epsilon^{(k)} + \frac{f''(\alpha)\left(\epsilon^{(k)}\right)^2}{2}$$

同様に$f'(x^{(k)})$もテイラー展開(こちらは$f'(\alpha) \neq 0$)

$$f'\left(x^{(k)}\right) \simeq f'(\alpha) + f''(\alpha)\epsilon^{(k)}$$

参考) 関数$f(x)$のa周りのテイラー展開：

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \cdots$$

収束性と初期値

ニュートン法の誤差収束(単根の場合)

$\epsilon^{(k+1)} = \epsilon^{(k)} - f(x^{(k)})/f'(x^{(k)})$にそれぞれ代入してやると:

$$\epsilon^{(k+1)} = \epsilon^{(k)} - \frac{f'(\alpha)\epsilon^{(k)} + \frac{1}{2}f''(\alpha)\left(\epsilon^{(k)}\right)^2}{f'(\alpha) + f''(\alpha)\epsilon^{(k)}} = \frac{f''(\alpha)\left(\epsilon^{(k)}\right)^2}{2(\underline{f'(\alpha) + f''(\alpha)\epsilon^{(k)}})}$$

真値α近くなら$f'(\alpha) \gg f''(\alpha)\epsilon^{(k)}$なので*

$$\epsilon^{(k+1)} = \frac{f''(\alpha)}{2f'(\alpha)}\left(\epsilon^{(k)}\right)^2$$

誤差が1反復処理毎に2乗となっているので,
単根の場合のニュートン法は**2次収束**

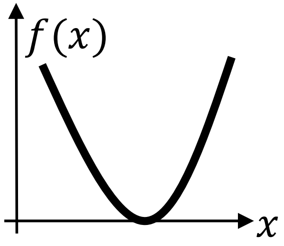
*真値近くなら$\epsilon^{(k)}$が小さくなるということが前提

収束性と初期値

ニュートン法の誤差収束(重根の場合)

重根の場合, $f(\alpha) = 0$であるとともに$f'(\alpha) = 0$でもあるので

$$\epsilon^{(k+1)} = \frac{f''(\alpha)\left(\epsilon^{(k)}\right)^2}{2\left(f'(\alpha) + f''(\alpha)\epsilon^{(k)}\right)} = \frac{1}{2}\epsilon^{(k)}$$



誤差が1反復処理毎に定数倍(< 1)となっているので,
重根の場合のニュートン法は**1次収束** (2分法と同じ)

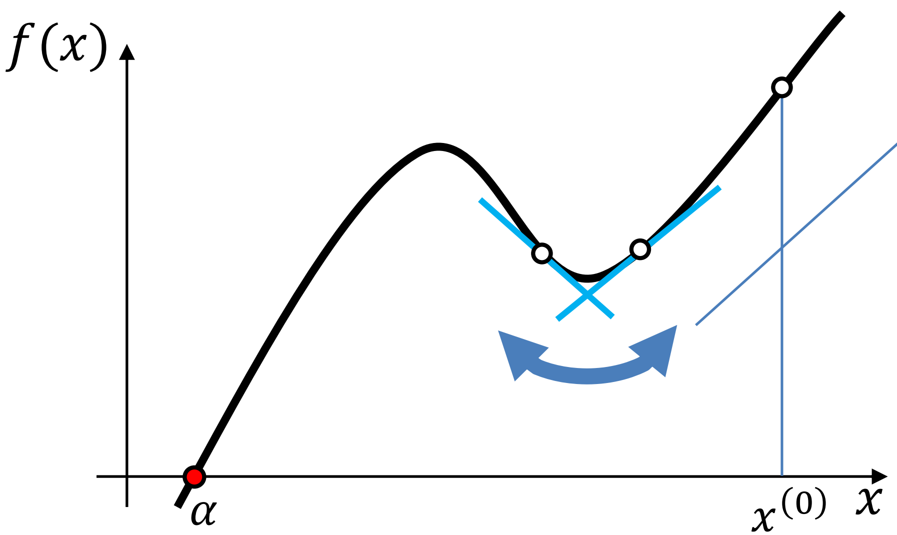
上の例は2重根の場合ですが, m重根($f'(\alpha)$から$f^{(m-1)}(\alpha)$が0で, $f^{(m)}(\alpha) \neq 0$)でも**1次収束**する*.
ただし, 収束率が$\frac{m-1}{m}$なのでm**が大きいほど収束は遅くなる**

*テイラー展開の高次項まで考えれば算出できる

収束性と初期値

ニュートン法が常に良いのか?

ニュートン法は収束が早く, 重根にも対応できるが,
初期値によっては**解に収束しない**場合もある



近似解$x^{(k)}$が進んだり戻ったり(振動)して, 真値にたどり着かない

この場合でも**2分法**なら初期範囲が問題なければ**確実に解が得られる**

2分法である程度解に近づいたならそれを**初期値として**
ニュートン法を使う

2分法, ニュートン法の拡張

- **Brentのアルゴリズム**
  2分法と割線法を**組み合わせ**て, ステップ毎に最適な方法を選択する + 直線ではなく**2次関数**で$x = 0$との交点を求める(逆2次補間)
  ⇒ 離散データに対してよく用いられる方法

- **Muller法, Laguerre(ラゲール)法, DKA法**など
  関数$f(x)$を**多項式に限定**することでより効率的に解くことができる方法
  (DKA法なら**初期値も計算で求められる**)

(DKA法はこの後説明します)

今回の講義内容

- 今日の問題
- 求根問題の数値計算での解き方
- 2分法
- ニュートン法
- 収束性と初期値
- **ホーナー法, DKA法**

多項式に対する求根問題

$f(x)$を**多項式(代数方程式)**に限定してみよう

n次多項式：

$$f(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-2}x^2 + a_{n-1}x + a_n$$

$f(x)$を計算するのに必要な**乗算回数**は$n + (n-1) + \cdots + 2 + 1 = \frac{n(n+1)}{2}$回で加算が$n$回

⇒ 乗算回数が多い(e.g. $n = 100$で乗算回数5050回)

⇩

乗算回数を減らすには？

ホーナー法

ホーナー法

ホーナー(Horner)法での多項式:

n次多項式：

$$f(x) = \Big(\big(\cdots\big((a_0x + a_1)x + a_2\big)x + \cdots + a_{n-2}\big)x + a_{n-1}\Big)x + a_n$$

多項式計算のコード例：

```
double func(double x, vector<double> a, int n){
    double f = a[0];
    for(int i = 1; i <= n; ++i){
        f = a[i]+f*x;
    }
    return f
}
```

$$\begin{aligned} a_0' &= a_0 \\ a_1' &= a_0'x + a_1 \\ a_2' &= a_1'x + a_2 \\ &\vdots \\ a_n' &= a_{n-1}'x + a_n \end{aligned}$$

と順番に計算*

$f(x)$を計算するのに必要な**乗算回数**はn回で加算もn回

⇒ 乗算回数が少ない(e.g. $n = 100$で乗算回数100回)

DKA法

$f(x)$を**多項式(代数方程式)**に限定した**求根問題**

n次多項式は重根なしならn**個の複素数解**を持つ*
$\Rightarrow$ n個の複素数解を**1度に計算する**方法を考えてみよう

多項式を変形する前に, 分かりやすくするために

$$P(z) = z^n + c_1 z^{n-1} + \cdots + c_{n-2} z^2 + c_{n-1} z + c_n = 0$$

と変数を変えます(複素解なので未知数をzにして, 変数を減らすために全体をa_0で割って$c_i = a_i / a_0$とした)

*実数も複素数の一部とする. また重根ありでも$\alpha_1 = \alpha_2$の2つの同じ値の解があるとするとn個の解となる

DKA法

代数方程式の解をα_i $(i = 0,1,\dots,n-1)$とすると多項式の因数分解によって:

$$P(z) = (z-\alpha_0)(z-\alpha_1)\cdots(z-\alpha_{n-1}) = \prod_{i=0}^{n-1}(z-\alpha_i)$$

ニュートン法の更新式にするために$P(z)$の微分*を計算しておく

$$P'(z) = \prod_{i=1}^{n-1}(z-\alpha_i) + \prod_{i=0,i\neq 1}^{n-1}(z-\alpha_i) + \cdots + \prod_{i=0,i\neq n-2}^{n-1}(z-\alpha_i) + \prod_{i=0}^{n-2}(z-\alpha_i)$$

$z=\alpha_0$ならば$z-\alpha_0=0$となるので上の式は第1項だけが残る. 同様にして$z=\alpha_j$**のとき第**j**項のみ残る**. これより, z_jが解に近い時, 以下の**近似式**が成り立つ

$$\boxed{P'(z_j) \simeq \prod_{i=0,i\neq j}^{n-1}(z_j - z_i)} \qquad (j = 0,1,\dots,n-1)$$

*積の微分($(fg)' = f'g + fg'$)を使う

DKA法

前ページ青枠の式をニュートン法の更新式に代入

$$z_j^{(k+1)} = z_j^{(k)} - \frac{P\left(z_j^{(k)}\right)}{P'\left(z_j^{(k)}\right)} = z_j^{(k)} - \frac{P\left(z_j^{(k)}\right)}{\prod_{i=0, i \neq j}^{n-1} \left(z_j^{(k)} - z_i^{(k)}\right)}$$

$$(j = 0,1, \dots, n-1)$$

この式(**DK式**)を使ってn個の解を求める方法を**デュラン・ケルナー(Durand-Kerner)法**(もしくはワイヤストラス法)*と呼ぶ

DK式はz_jが解α_jに近いことを前提としているので初期値$z_j^{(0)}$の選び方が重要

⇨ **アバースの初期値**

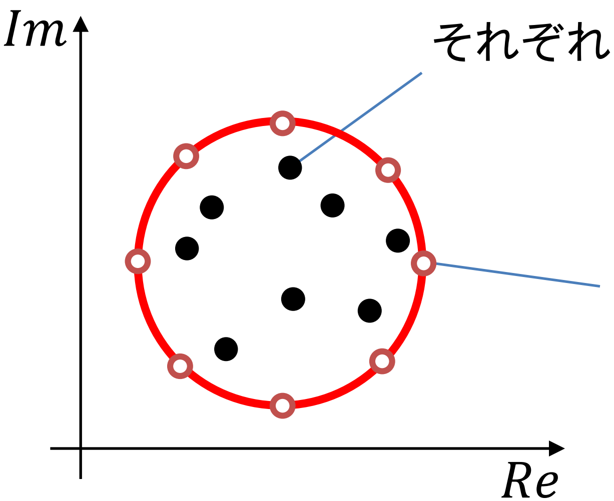
*ワイヤストラスが1891年に開発していたけど, それを知らずにデュランとケルナーがそれぞれ1960年と1966年に独自開発

DKA法

アバース(Aberth)の初期値*

$$z_j^{(0)} = \underbrace{-\frac{c_1}{n}}_{\text{円の中心}} + r \exp\left\{ i \left(\underbrace{\frac{2\pi}{n}}_{\text{初期値の配置間隔(rad)}} j + \frac{\pi}{2n} \right) \right\}$$

$j = 0,1,\ldots,n-1$, rは複素平面上でn個の解をすべて包むような円の半径, iはインデックスではなくここでは虚数($\sqrt{-1}$)



それぞれの黒点は複素解$\alpha_j = x_j + iy_j$をプロットしたもの

全点を含む半径rの円上に等間隔に初期値設定

- 円の中心 = 解の重心
  $$\frac{\alpha_0+\alpha_1+\cdots+\alpha_{n-1}}{n} = -\frac{c_1}{n}$$
- 配置間隔 : $\frac{2\pi}{n}$, 配置オフセット: $\frac{\pi}{2n}$ ($\frac{3}{2n}$とする場合も)

*式の導出についてはこちらのリンク参考

DKA法

半径rはどうやって決める?

アバースの方法による半径rの計算方法*

$z = w - c_1/n$と置くと $P(w) = w^n + c_2' w^{n-2} + \cdots + c_{n-1}' w + c_n' = 0$
$c_i' \neq 0$ならば以下の方程式が成り立つ

$$S(w) = w^n - |c_2'| w^{n-2} - \cdots - |c_{n-1}'| w - |c_n'| = 0$$

注) z^{n-1}の係数c_1を基準にwを決めてるので, w^{n-1}の項だけない

$S(w) = 0$は1個の根rをもち, $P(z)$の解は半径r内の閉円板に属する(Smithの定理)**

$S(w)$の係数c_i'はホーナー法を応用した組立除法で求められ, その根は2分法やニュートン法を用いれば求められる

(例えば, $r^{(0)} = \max_{k=2,\ldots,n} (m|c_k|/|c_0|)^{\frac{1}{k}}$から探索($m$は係数$c_k$で0でないものの個数)***)

*O. Aberth, "Iteration methods for finding all zeros of a polynomial simultaneously", Math. Comput. 27, pp.339-344, 1973.
**これによって半径がなぜ求まるかは, 山本哲朗,数値解析入門,サイエンス社 の付録Aあたりが参考になる
***小澤一文,"Durand-Kerner法の効率的な初期値の簡単な設定方法",日本応用数理学会論文誌, 3(4), pp451-464, 1993.

DKA法

DK式とアバース(Aberth)の初期値の組み合わせ

Aberthの初期値：
$$z_j^{(0)} = -\frac{1}{n}\frac{c_1}{c_0} + r\exp\left\{i\left(\frac{2\pi}{n}(j-1) + \frac{\pi}{2n}\right)\right\}$$

DK式：
$$z_j^{(k+1)} = z_j^{(k)} - \frac{P\left(z_j^{(k)}\right)}{\prod_{i=0, i\neq j}^{n-1}\left(z_j^{(k)} - z_i^{(k)}\right)}$$

$$(j = 0,1,\ldots,n-1)$$

Aberthの初期値$z_j^{(0)}$を使って
DK式で$z_j^{(k)}$を更新していく方法

⇒ DKA法

DKA法

アバースの初期値のコード例

```
// 半径算出のための方程式の係数
 (Appendix参照)
// 多項式S(w)の係数
vector<double> b(cd.size());
b[0] = abs(cd[0]);
for(int i = 1; i <= n; ++i) b[i] = -abs(cd[i]);

// Aberthの初期値の半径をニュートン法で算出
double r = r0;
newton(b, n, r, max_iter, eps);

// Aberthの初期値
complexf zc = -c[1]/(c[0]*(double)n);
for(int j = 0; j < n; ++j){
    double theta = (2*RX_PI/(double)n)*j+RX_PI/(2.0*n);
    z[j] = zc+r*complexf(cos(theta), sin(theta));
}
```

$S(w) = w^n - |c_2'|w^{n-2} - \cdots - |c_n'|$ の係数計算

ニュートン法(多項式の係数を引数にしたもの)で$S(w)$の根(半径r)を求める

complex型についてはAppendix参照

z_jの初期値の計算. オイラーの公式から $\exp(i\theta) = \cos\theta + i\sin\theta$として計算している

DKA法

DK式によるzの更新

$$z_j^{(k+1)} = z_j^{(k)} - \frac{P\left(z_j^{(k)}\right)}{\prod_{i=0, i\neq j}^{n-1}\left(z_j^{(k)} - z_i^{(k)}\right)}$$

```
vector<complexf> zp;
complexf f, df;
int k;
for(k = 0; k < max_iter; ++k){
    zp = z;

    // DK式の計算
    for(int j = 0; j < n; ++j){
        f = func(z[j], c, n);
        df = c[0];
        for(int i = 0; i < n; ++i){
            if(i != j) df *= zp[j]-zp[i];
        }
        z[j] = zp[j]-f/df;
    }

    // 誤差の算出と収束判定
     (省略)
}
```

DK式のために$P(z_j)$と$P'(z_j)$を計算

DK式の計算

収束判定は$\left|P\left(z^{(k)}\right)\right| < \varepsilon$で行う

DKA法

DKA法の実行結果例1

$$f(x) = 2x^5 + 5x^3 + 3x + 1 = 0$$ 実数解 $x \fallingdotseq -0.290911$

許容誤差$\varepsilon = 1 \times 10^{-6}$

```
r = 1.75488
x0(0) = 1.66899 + 0.542287i,
x1(0) = 1.07455e-16 + 1.75488i,
x2(0) = -1.66899 + 0.542287i,
x3(0) = -1.03149 + -1.41973i,
x4(0) = 1.03149 + -1.41973i

solutions :
x0 = 0.287248 + 0.938484i
x1 = -0.141792 + 1.32822i
x2 = -0.290911
x3 = -0.141792 + -1.32822i
x4 = 0.287248 + -0.938484i

iter = 7, eps = 2.05905e-12
```

アバースの方法による
半径と初期値の計算結果

実数解だけでなく
複素解もすべて求まっている

7回の反復で処理終了

DKA法

DKA法の実行結果例2

$$f(x) = x^5 - 3x^4 + 9x^3 - 37x^2 + 80x - 50 = 0$$

解 $x = 1, 2 \pm i, -1 \pm 3i$

許容誤差$\varepsilon = 1 \times 10^{-6}$

```
r = 3.87418
x0(0) = 4.28456 + 1.19719i,
x1(0) = 0.6 + 3.87418i,
x2(0) = -3.08456 + 1.19719i,
x3(0) = -1.67719 + -3.13428i,
x4(0) = 2.87719 + -3.13428i

solutions :
x0 =  2 + 1i   --> f = -8.94573e-12 (OK)
x1 = -1 + 3i   --> f = 0 (OK)
x2 =  1        --> f = -2.42739e-12 (OK)
x3 = -1 + -3i  --> f = 1.7053e-13 (OK)
x4 =  2 + -1i  --> f = -1.63425e-13 (OK)

iter = 9, eps = 1.18566e-11
```

解の正しさは$f(x)$に得られた解を代入してみれば分かる(0に近いほどよい)

9回の反復で処理終了

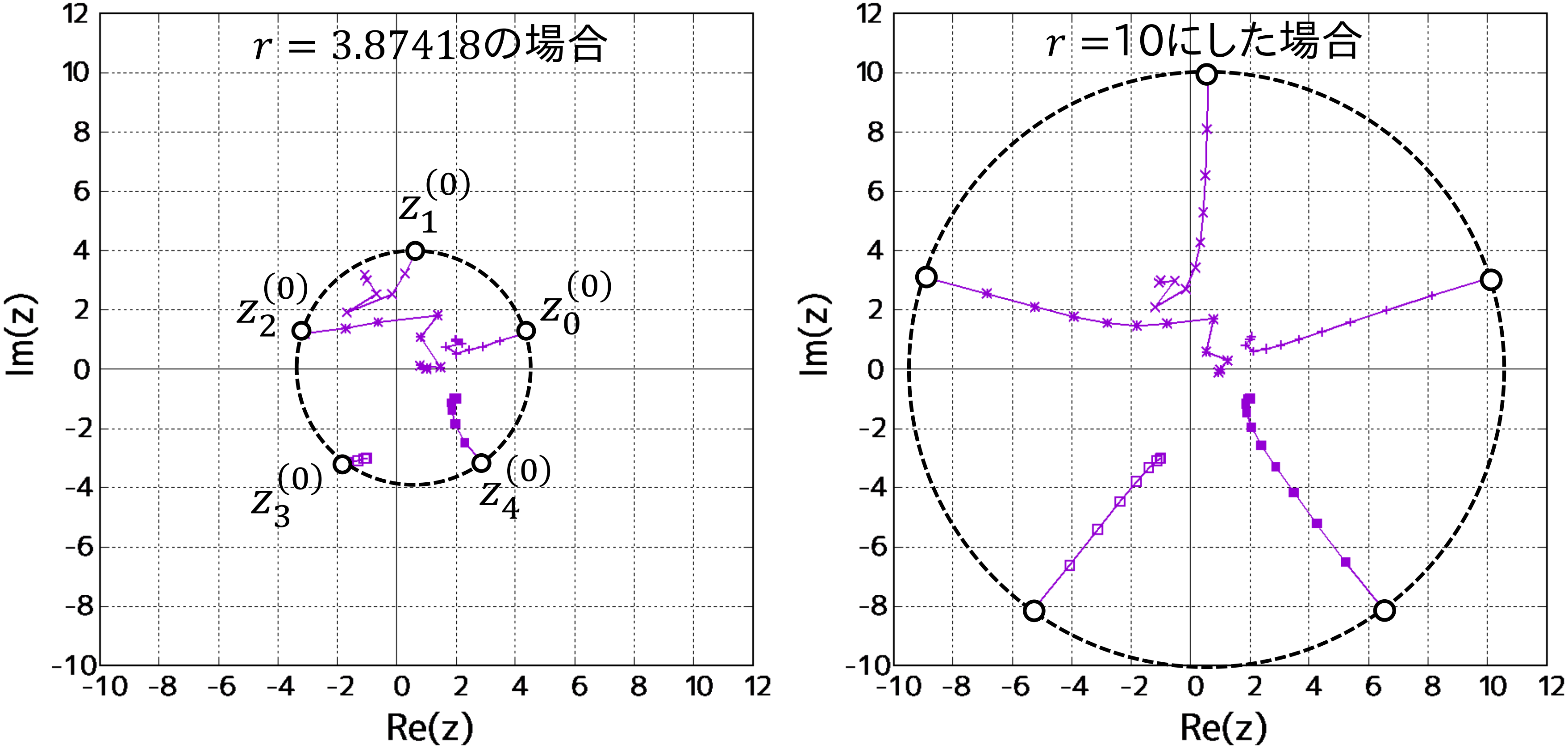
DKA法

初期値からの移動を確認してみよう

$$f(x) = x^5 - 3x^4 + 9x^3 - 37x^2 + 80x - 50 = 0$$

解 $x = 1, 2 \pm i, -1 \pm 3i$

r = 3.87418 アバースの初期値の半径



講義のまとめ

- 今日の問題 : $f(x) = 0$となるxは?
- 求根問題の数値計算での解き方
  - 反復計算で解けるけど効率を上げるにはどうすべきか
- 2分法, ニュートン法
  - 区間を分割する方法と微分を用いる方法
- 収束性と初期値
  - 重根, 縮小写像の定理, 1次収束と2次収束
- ホーナー法, DKA法
  - 多項式の場合の数値計算法

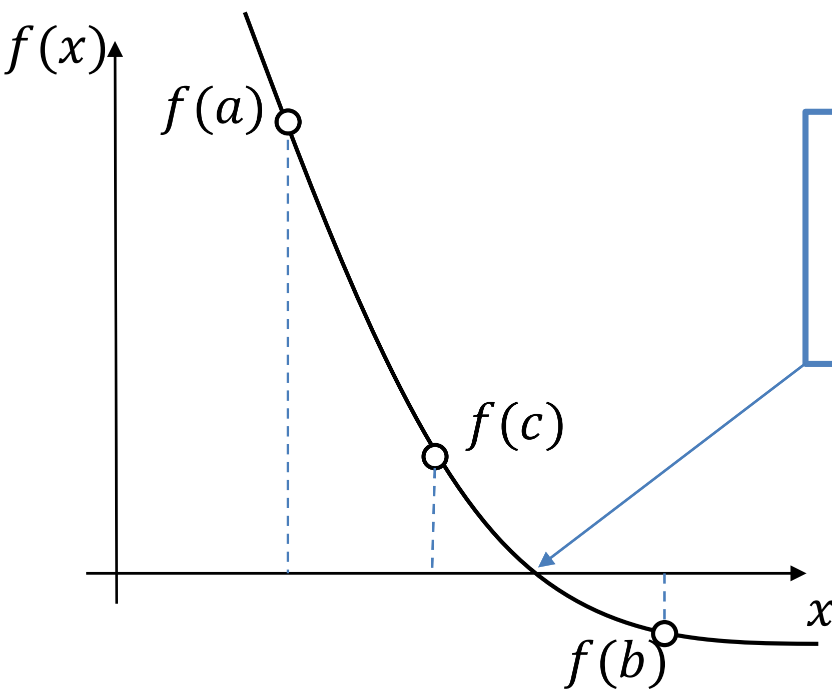
Appendix

(以降のページは補足資料です)

中間値の定理

中間値の定理

閉区間$[a,b]$で連続な関数$f(x)$について，
$f(a) \neq f(b)$ならば，$f(a)$と$f(b)$の間の任意の数kに対して，$f(c)=k$となる$c\ (a<c<b)$が存在する



$f(a) > 0$で$f(b) < 0$なら
$[a,b]$内に$f(c)=0$となる
点cが存在する

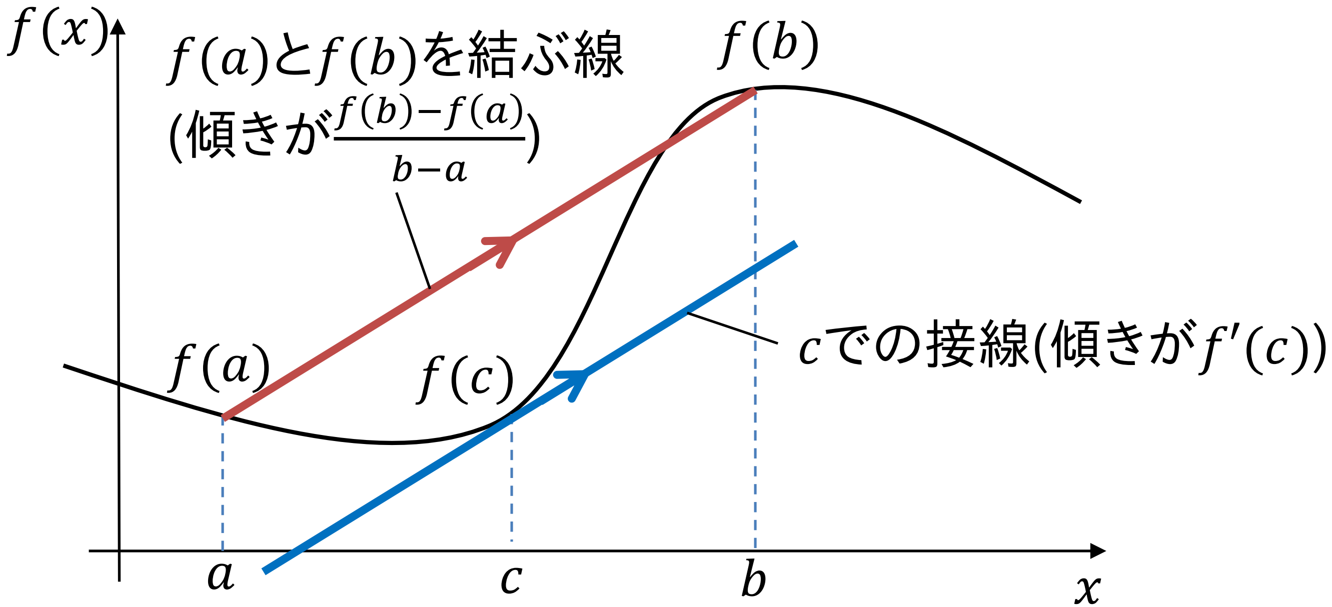
平均値の定理

平均値の定理

区間$[a, b]$内で微分可能な関数$f(x)$について，

$$\frac{f(b) - f(a)}{b - a} = f'(c)$$

となるcが，aとbの間に少なくとも1つは存在する



C++のcomplex型について

C++では複素数を扱うための型が用意されている

```
#include <complex>// 複素数

complex<double> x;
x = complex<double>(1, 2); // 1+2iが代入される
cout << x.real() << " + " << x.imag() << "i" << endl;
complex<double> y, z;
y = complex<double>(2, 3)
z = x + y; // 複素数同士の四則演算も可能
```

なおサンプルプログラムでは以下のように
double型の複素数を定義して使っている

```
typedef complex<double> complexf; // double型の複素数を定義
```

組立除法による係数計算

組立除法：多項式を1次式で割ったときの商と余りを求める方法

例) $f(x) = x^3 + 2x^2 + 3x + 4$ を $(x - 2)$ で割ったら？

この2を持ってくる

$f(x)$の係数を並べる

1	2	3	4	2
コピー	+	+	+	
	2	8	22	
	×2	×2	×2	
1	4	11	26	

商となる$(n-1)$次多項式の係数　余り

青矢印：右上の数字を掛ける
緑矢印：上2つの数字を足す
という意味

$$x^3 + 2x^2 + 3x + 4 = (x-2)(x^2 + 4x + 11) + 26$$

組立除法による係数計算

組立除法を使って$(x-a)$に関する多項式に変換

例) $f(x)=x^3+2x^2+3x+4$ を $(x-2)$ で割ったら,

$$(x-2)(x^2+4x+11)+26$$

$\Rightarrow x^2+4x+11$を更に$(x-2)$で割ったら?

$$(x-2)\big((x-2)(x+6)+23\big)+26$$

$(x+6)$を$(x-2)+8$とすると

$$(x-2)\big((x-2)((x-2)+8)+23\big)+26$$

$(x-2)$を残したまま展開

$$(x-2)^3+8(x-2)^2+23(x-2)+26$$

$(x-2)$に関する多項式に分解できた!

$P(z)$を$z+\frac{c_1}{n}=w$で割っていけば同様にして$\hat{P}(w)$を求められる

組立除法による係数計算

組立除法の計算コード例

```
void horner(const vector<double> &a, double b, vector<double> &c, double &rm, int n)
{
    rm = a[0]; // 最終的に余りになる
    c.resize(n);
    for(int i = 1; i < n+1; ++i){
        c[i-1] = rm;         // 結果の格納
        rm *= b; rm += a[i]; // 組立除法
    }
}
```

$(x-b)$で割るとして，$\times b$の処理と上2つの数字を足す処理

組立除法による係数c_i'の計算コード例

```
// 半径算出のための方程式の係数
vector<complexf> cd(n+1, 1.0); // 係数c'
double c1n = -c[1].real()/n; cd[0] = c[0];
double rm; // 組立除法での余り
vector<double> a(n+1), tmp(n); // 係数格納用の一時的な変数
for(int i = 0; i <= n; ++i) a[i] = c[i].real();
// zの多項式をz+c1/nで割っていくことでwの多項式の係数を求める
for(int i = n; i > 1; --i){
    horner(a, c1n, tmp, rm, i);
    cd[i] = rm; a = tmp;
}
cd[1] = a[1]+c1n;
```

$(z+c_1/n)$の設定と $c_0' = c_0$

反復計算用に別の配列a[ ]に係数を入れておく

組立除法で出てきた余りが$\hat{P}(w)$の係数になる