

数値解析 テスト対策

2021年8月2日

目次

1	1 変数の方程式	3
1.1	二分法	3
1.2	ニュートン法	3
1.3	例題	3
2	標本点から関数を求める	4
2.1	ラグランジュ補間	4
2.2	スプライン補間	4
2.3	最小二乗法	4
3	連立方程式	5
3.1	ガウスの消去法	5
3.2	LU 分解	5
3.3	ヤコビ行列	5
3.4	ガウス-ザイデル法	5
3.5	SOR 法	5
4	積分計算	6
4.1	台形則	6
4.2	シンプソン則	6
4.3	ロンバーグ積分法	6
4.4	モンテカルロ積分法	6
5	微分方程式	7
5.1	オイラー法	7
5.2	修正オイラー法	7
5.3	ルンゲクッタ法	7

1 1変数の方程式

1.1 二分法

解をとおき、挟み撃ちで実数上で検討をつけていく方法である。中間値の定理を使用

中間値の定理: $f(a) < 0, f(b) > 0$ となる a, b が存在すれば $f(\quad) = 0$ となるような が a と b の間に少なくとも 1 つ存在する。

中間値である が 0 になればこの数値計算は終了し、答えになる。しかし、実際のコンピュータの計算では 0 を求めるまでにたくさんの時間が必要になる。そのため、 $(a+b)/2$ が自分で決めた誤差 (0.00001 など) より小さければ収束判定を行う。

1.2 ニュートン法

解をとおき の近くで初期値 x_0 をとる。接戦を引き $y = 0$ との交点を x_1 とする。そして、 x_1 の接戦を引き…以下ループ。

二分法では、中間値の定理を用いて挟み撃ちをしていたが、ニュートン法はテイラー展開の近似式を用いて解を近似していく。誤差と残差によって収束判定を行うことができるが、一般的には誤差による収束判定を行う。(誤差 := x の解と点の x 座標の差、残差 := $y = 0$ と点の y 座標の差)

ニュートン法には、解を求められない場合がある。

ニュートン法の方が高速

1.3 例題

$x^2 - 1 - \sin x = 0$ の数値解を求める方法について示せ。

2 標本点から関数を求める

2.1 ラグランジュ補間

ラグランジュは与えられた点をすべて通る曲線を生成する方法である。似た方法にスプライン補間がある。ラグランジュ補間は、標本点の数が少ない上手くいく。しかし、標本点の数が多くなる場合には両端の標本点付近で推定値が振動的になり近似精度が著しく悪くなる。(ルンゲの現象)

2.2 スプライン補間

ラグランジュ補間と同様の与えられた点をすべて通る曲線を生成する方法である。ラグランジュ補間と違う点は、ルンゲの現象が起きないように区間の境目を滑らかにしていることである。スプライン補間になる条件は 2 つある。

1. $y = S(x)$ は連続であり点をすべて通る。
2. 区間の境目における曲線の滑らかさを与える- \therefore 1 階微分係数と 2 回微分係数が連続である。

2.3 最小二乗法

標本点からそれに近い直線を出すときに使用される方法。二乗誤差の総和を最小にするように $y = ax + b$ の a と b を定めることを考える。すべての標本点を通るようにすることができます、近似の直線を求めることができない。

3 連立方程式

3.1 ガウスの消去法

行基本変形ともいわれる行列の計算方法のことである。数値解析の際は、解くのに時間がかかるため良い方法だと言えない。

3.2 LU 分解

ガウスの消去法よりも効率が良い連立方程式の求め方。数値解析の際は、解くのに時間がかかるため良い方法だと言えない。

数値解析の場合に用いられる反復法を用いた連立方程式の求め方は以下の 3 つである。いずれも直接解を求めるのではなく、解を近似していく方法である。

3.3 ヤコビ行列

ヤコビ行列は連立方程式の解を出す解法の例の一つである。最初に、適当な初期値を取って近似させていく。

3.4 ガウス-ザイデル法

ガウス-ザイデル法はヤコビ行列よりもほぼ 2 倍の速さで解が収束していく。その理由は、連立方程式を上手く活用することにより、初期値を取る数が減りそれぞれの値 (x_1, x_2 など) が 2 倍の速さで近似していくためである。

3.5 SOR 法

ヤコビ行列とガウス-ザイデル法は適当な初期値を取りそこから収束させるのに対し、SOR 法はこの初期値に加え加速係数を追加することにより、さらに高速に解を収束することができる数値計算である。

4 積分計算

4.1 台形則

1 次のラグランジュ補間多項式を利用している。そのため誤差がある。 $(1/n^2)$

4.2 シンプソン則

2 次のラグランジュ補間多項式で近似。誤差 $(1/n^4)$

シンプソン則と台形則の関係：台形則 (T_n) 、シンプソン則 (S_n) とすると、

$$S_n + 1 = 4/3T_n + 1 - 1/3T_n$$

4.3 ロンバーグ積分法

上記の 2 つよりはやく誤差が小さくなる積分法である。

今まで述べた積分法だとより複雑な積分になったときに求めるのは難しくなる。そのため、複雑な関数の積分ができるようにしたもののが以下の積分法である。

4.4 モンテカルロ積分法

複雑な関数の積分計算が可能な積分法である。精度はサンプル数の平方根のため、誤差を $1/10$ にしたければサンプル数を 100 倍に増やすなければならない。

5 微分方程式

5.1 オイラー法

テイラー展開を利用した数値解析である。ステップ幅 h の 2 次以降の項はカットするため誤差は h^2 の位になる。

5.2 修正オイラー法

同じくテイラー展開を利用した数値解析である。誤差は h^3 の位になる。

5.3 ルングクッタ法

オイラー法、修正オイラー法よりも一層正確な計算ができる。誤差は h^5 の位になる。(h^4 までは正確な計算ができる) 一般的な微分方程式の数値解法といった場合はルングクッタ法を指している場合が多い。

第3章

方程式の解

■人類の命運をかけた方程式

v を速度 (ft/s) とするとき次の方程式 $g(v) = 0$ を解いて、その解が $v > 40 ft/s$ となるか否かについて明らかにせよ。

$$g(v) = v + d + a \log(1 - (v/a)) = 0 \quad (3.1)$$

ここで、 d, a はそれぞれある定数。

■目標

x の任意の関数を $f(x)$ としたとき

$$f(x) = 0 \quad (3.2)$$

を満たす解をもとめるための数値計算法の方法と原理を覚える。

■学習のポイント

- 数値計算における重要な要素
 - 計算量
 - 収束判定
- 2分法、ニュートン法
 - 計算原理
 - 長所と欠点
 - プログラム

例 $x^2 = 11$ を考える (教科書 p22) : 方程式 $f(x) = x^2 - 11 = 0$ を解く

方策 1

～解の公式がないと使えない方法

方策 2 解を α としておき, 挟み撃ちで見当をつけていく

3 と 4 の間を狭めていけば, おおよそ見つけられそう

疑問 方策 2 は解の公式がなくても使えそうであるが, 一般性はあるのか?

$$3 < a < 4 \rightarrow \frac{3+4}{2} = 3.5 \quad (3.5)^2 = 12.25$$

$$\therefore 3 < a < 3.5 \rightarrow \frac{3+3.5}{2} = 3.25 \quad (3.25)^2 = 10.56 \cdots$$

$$\therefore 3.25 < a < 3.5 \rightarrow \frac{3.25+3.5}{2} = 3.375 \quad (3.375)^2 = 11.39 \cdots$$

$$\therefore 3.25 < a < 3.375 \rightarrow \frac{3.25+3.375}{2} = 3.3125 \quad (3.3125)^2 = 10.97 \cdots$$

$$\therefore 3.3125 < a < 3.375 \rightarrow \cdots$$

3.1 2分法 (bisection method)

方程式の解は複数個あるかもしれないが、方策2でとにかく1つ解を求めることができる根拠となるのが、あの定理。

定理 3.1.1

もし、 $f(a) < 0, f(b) > 0$ となる a, b が存在すれば、

2分法のアルゴリズム（暫定版）

- (1) $f(a) < 0, f(b) > 0$ を満たすように変数 a, b の値を設定する。
- (2) 以下繰り返し

$$f_c := f(c)$$

もし

- $f_c > 0$ ならば
- $f_c < 0$ ならば
- $f_c = 0$ ならば ステップ(3)へ

- (3) c を答えとする

上記アルゴリズムでは、 $f_c = 0$ となることは、ほぼありえないので繰り返しが終了しない。そこでほぼ、必要な解の桁数までもとめることができたとして計算を打ち切るための条件を設ける。

収束判定条件

$c = (a + b)/2$ としたとき、ある正数 ε を考え、となれば、計算
を打ち切り、解を c とする。

2分法のアルゴリズム（収束判定条件付き）

- (1) $f(a) < 0, f(b) > 0$ を満たすように変数 a, b の値を設定する。正数 ε を設定する。
- (2) 以下繰り返し

$$c := (a + b)/2$$

$$f_c := f(c)$$

もし

- $f_c > 0$ ならば $b := c$
- $f_c < 0$ ならば $a := c$
- $f_c = 0$ ならば ステップ(3)へ

- (3) c を答えとする

3.1.1 2分法の計算量

- 2分法の計算の手間：ステップ(2)での $f(c)$ の計算回数に依存
- ステップ(2)を1回実行すると区間 (a, b) の幅は半分になる
- ゆえに計算を終了するまでに要する $f(c)$ の計算回数 N を用いて

$$\frac{|a - b|}{2^{N+1}} < \varepsilon \quad (3.3)$$

- 上記を満たす最小の N は

$$(3.4)$$

•

2分法は、次回説明するニュートン法よりも速さの点で劣るが、1回の計算ごとに確実に解に接近する信頼性が高い方法である。

考察 2分法で $f(x) = (x - 1)^2 = 0$ の解が求められるかどうか考察せよ

3.2 ニュートン法

2分法は安全確実な方法だが、収束までに時間がかかる。より高速で、広く使われているニュートン法について学ぶ。

3.2.1 ニュートン法の原理

- 1) 初期値 解 α の付近で初期値 x_0 をとる
- 2) 接線 点 $(x_0, f(x_0))$ での接線を引く
- 3) 交点 この接線と x 軸との交点を x_1 とする
- 4) 繰り返し 再び $(x_1, f(x_1))$ における接線を求め、その接線と x 軸との交点を x_2 と求め、以降これを繰り返す。

x_{n+1} の求め方 n 番目に求められた交点 x_n の点 $(x_n, f(x_n))$ の接線の式は

(3.5)

この式の $y = 0$ を求めれば、それが x_{n+1} となるので

$$x_{n+1} = \boxed{\dots} \quad (3.6)$$

この方法の原理は？ 区間 (a, b) で $x = \alpha$ が方程式 $f(x) = 0$ の解と仮定。いま、近似解 x_n の近傍で

$$0 = f(\alpha) = f(x_n) + \dots \quad (3.7)$$

$\varepsilon_n = \alpha - x_n$ が十分小さいとすれば、第 3 項は省略可能で、 α について解くと

$$\alpha \cong \boxed{\dots} \quad (3.8)$$

すなわち、ニュートン法はテイラー展開の近似式を使って、近似解を求めていく方法である。

3.2.2 収束判定条件

2 分法と違って、ニュートン法では現在の値 x_n がどれだけ、解に近づいているか知る方法がない。どこかで、十分に収束したとみなせるところで計算を打ち切り、そのときの x_n を近似解とする。収束判定の基準には、誤差によるものと残差によるものがある。

- 誤差は $|x_n - \alpha|$ で表される。
- 残差は 0 になるべき $f(x_n)$ と 0との差 $|f(x_n)|$ で表される。
- 残差による収束判定：許容誤差 $\delta > 0$ を決めておき、 $|f(x_n)| < \delta$ となったら収束したとみなす。
 - 梱落ちによる不正確性や、残差が小さいから誤差も小さいという保証はないため、一般的には誤差基準を用いる。

- 誤差による収束判定：事前に許容誤差 $\epsilon > 0$ を決めておき、 $|x_n - \alpha| < \epsilon$ なつたら収束としたい。
 - 実際には、 α は未知： x_{n-1} で代用、 $|x_n - x_{n-1}| < \epsilon$ で収束したとみなす。
 - しかし「小数点以下 5 桁まで正確に求めよ」と言わされた場合には、この方法では ϵ をどのように設定すれば良いかわからない。
 - これは、 $|x_{n+1} - x_n| < 10^{-5}|x_{n+1}|$ とすれば良い。

表 3.1 x_{n+1} の収束判定

n	x_{n+1}	x_n	$x_{n+1} - x_n$	$\frac{x_{n+1} - x_n}{x_{n+1}}$
0	4.000000000			
1	3.197252957	4.000000000	0.802747043	0.251073986
2	3.141865232	3.197252957	0.055387725	0.017628931
3	3.141598596	3.141865232	0.000266636	0.000084873
4	3.141592678	3.141598596	0.000005918	0.000001884
5	3.141592653	3.141592678	0.000000025	0.000000008

- x_{n+1} と x_n の差が N 桁一致した時点で計算を終了する
 - ある正数 $\epsilon (= 10^{-N})$ を設定し

$$\left| \frac{x_{n+1} - x_n}{x_{n+1}} \right| < \epsilon \quad \text{または} \quad (3.9)$$

3.2.3 ニュートン法のアルゴリズム

(1) 初期値 x を設定, ε を設定

(2) 以下繰り返し

交点の計算

$$x_{new} := x - \frac{f(x)}{f'(x)} \quad (3.10)$$

収束判定条件

ならば、ステップ (3) に移る (3.11)

x の更新

$$x := x_{new} \quad (3.12)$$

(3) x_{new} を答えとする

3.2.4 ニュートン法の計算例

$$f(x) = e^{-x} - x^2 = 0 \quad (3.13)$$

2-2 =			
表 2-1 $e^{-x} - x^2 = 0$ の根を 2 分法で 計算した結果			
n	c	n	c
1	0.500000	10	0.7041015
2	0.750000	11	0.7036132
3	0.625000	12	0.7033691
4	0.687500	13	0.7034912
5	0.7187500	14	0.7034301
6	0.7031250	15	0.7034606
7	0.7109375	16	0.7034759
8	0.7070312	17	0.7034683
9	0.7050781	18	0.7034645
真の根は $0.7034674\cdots$			

2-3 =	
表 2-3 $e^{-x} - x^2 = 0$ の根をニュ ートン法で計算した結果	
n	x_n
0	1
1	0.733043605245445
2	0.703807786324133
3	0.703467468331797
4	0.703467422498392
真の根は $0.703467422498391\cdots$	

3.2.5 ニュートン法の計算量

1回の繰り返し計算に $f(x)$ と $f'(x)$ の計算, 合計2回の関数計算がある
 ある計算で7桁正しい計算を得るために, 2分法では23回の計算 ($\varepsilon < 10^{-7}$), ニュートン法では6回, ニュートン法はこの計算例では, 4倍高速
 ニュートン法が高速に解く理由 解 α と x_n との誤差を $\varepsilon_n = x_n - \alpha$ とおく

$$\varepsilon_{n+1} = x_{n+1} - \alpha = x_n - \frac{f(x_n)}{f'(x_n)} - \alpha = \varepsilon_n - \frac{f(x_n)}{f'(x_n)} \quad (3.14)$$

$x_n = \alpha + (\varepsilon_n - \alpha) = \alpha + \varepsilon_n$ と書き換える, デイラー展開より

$$f(x_n) = f(\alpha) + \varepsilon_n f'(\alpha) + \frac{\varepsilon_n^2}{2} f''(\alpha) + \frac{\varepsilon_n^3}{6} f'''(\xi_1) \quad (3.15)$$

$$f'(x_n) = f'(\alpha) + \varepsilon_n f''(\alpha) + \frac{\varepsilon_n^2}{2} f'''(\xi_2) \quad (3.16)$$

ξ_1, ξ_2 はそれぞれ x_n と α の間の数

上記を式(3.14)に代入し, $f(\alpha) = 0$ を考慮

$$\varepsilon_{n+1} = \varepsilon_n - \frac{\varepsilon_n f'(\alpha) + \frac{\varepsilon_n^2}{2} f''(\alpha) + \frac{\varepsilon_n^3}{6} f'''(\xi_1)}{f'(\alpha) + \varepsilon_n f''(\alpha) + \frac{\varepsilon_n^2}{2} f'''(\xi_2)} \quad (3.17)$$

$$= \frac{\left(\varepsilon_n f'(\alpha) + \varepsilon_n^2 f''(\alpha) + \frac{\varepsilon_n^3}{2} f'''(\xi_2) \right) - \left(\varepsilon_n f'(\alpha) + \frac{\varepsilon_n^2}{2} f''(\alpha) + \frac{\varepsilon_n^3}{6} f'''(\xi_1) \right)}{f'(\alpha) + \varepsilon_n f''(\alpha) + \frac{\varepsilon_n^2}{2} f'''(\xi_2)} \\ = \quad (3.18)$$

$$\simeq \quad (3.19)$$

すなわち, 誤差を指数関数的に減少させる. (x_n が小数点以下 p 桁正しい近似値であったとき, x_{n+1} は少なくとも約 $2p$ 桁正確くなっている)

3.2.6 ニュートン法の短所

3.2.7 演習

- $e^{-x} - x^2 = 0$ の数値解を求めるニュートン法のプログラムを作成し、その計算結果のスクリーンショットを取り、考察を加えて、manaba の数値解析レポート 2) へ提出のこと。

3.2.8 次回への予習

体積が $153m^3$ となる立方体の辺の長さが知りたい。この辺の長さを求めるプログラムを作成して、長さを求めよ。

- 求める体積を p として、辺の長さを x_n とする。
- 暫定的に幅、奥行きの長さ x_n 、高さ $\frac{p}{x_n^2}$ の直方体によって、この立方体を近似する。
- $x_n = \frac{p}{x_n^2}$ ならば、この直方体は立方体になるので、解は x_n となる。
- そうでない時には、幅、奥行きと高さには差があるので、{ 幅、奥行き、高さの平均値 } を新たな幅 x_{n+1} として直方体を更新する

$$x_{n+1} = \frac{1}{3} \left(x_n + x_n + \frac{p}{x_n^2} \right) \quad (3.20)$$

- 方程式 $x^3 - p = 0$ をニュートン法で解くときの漸化式が上式と一致することを確認、プログラムを作成せよ。

体積 $p (= 153)$ となる立方体の辺の長さを求めるプログラム例

```
# coding: UTF-8
from math import exp

p=153

def f(x):
    return x**3-p

def f_prime(x):
    return 3*x*x

def Newton(ini,err):
    x_n=ini
    x_n_scc=0
    count=0
    while True:
        count=count+1
        x_n_scc=x_n-f(x_n)/f_prime(x_n)
        print(\n
              count," Slolution:",x_n_scc,\n)

        if abs(x_n_scc-x_n)/x_n_scc<err:
            break
        x_n=x_n_scc
    print(\n
          "Slolution:",x_n_scc,\n
          "Function Value:",f(x_n_scc),"\n"
          "Iteration:",count)

if __name__ == '__main__':
    Newton(4.0, 0.000001)
```

演習 3.2.1 放射性廃棄物の海洋投棄は安全か？

アメリカ原子力委員会は、数年にわたって濃縮した放射性廃棄物をドラム缶に密封し、推進 300 ft (約 100m) の地点に海洋投棄していた。この処置に危機感を抱いた生態学者や科学者が問い合わせたところ、原子力委員会は放射能漏れを起こすことは決してないと保証した。ドラム缶の大規模な試験の結果も、原子力委員会の主張を裏打ちするものとなった。しかし、技術者の中から、海底に落ちた時の衝撃から、ドラム缶に亀裂が生じるのではないかとの質問があった。原子力委員会の回答は「ありえない」というものだった。技術者たちは「実験してみればわかる」とし、数多くの実験を行った後、秒速 40ft を超える速度で海底に衝突するとドラム缶に亀裂が生じることがわかった。ここでの問題は、海底に衝突する時点のドラム缶の速度を計算し、秒速 40ft を超えるか否かを明らかにすることである。

海中を落下していくドラム缶の速度は、ニュートン力学から微分方程式を立てることによってモデル化できる。

(微分方程式から、速度を計算するための式の導出過程は、ここでは省略する)

モデル化を通じて、ドラム缶の衝突速度は、次の方程式を満たすことがわかった。

$$g(v) = v + \frac{300cg}{W} + \frac{W-B}{c} \log \left[\frac{W-B-cv}{W-B} \right] = 0 \quad (3.21)$$

ここで

c	0.08	正の定数
g	32.2	重力加速度 (ft/s^2)
W	527.436	ドラム缶が下に引っ張られる重さ
B	470.327	ドラム缶位働く浮力

式 (3.21)において、 $a = (W - B)/c$, $d = 300cg/W$ とすると以下のように単純化できる。

$$g(v) = v + d + a \log(1 - (v/a)) = 0 \quad (3.22)$$

上式の問題に対して、ニュートン法を用いると

$$v_{n+1} = v_n - \frac{g(v_n)}{g'(v_n)} = v_n - \frac{v_n + d + a \log(1 - (v_n/a))}{-\frac{v_n/a}{1-v_n/a}} \quad (3.23)$$

$$= v_n + \frac{a - v_n}{v_n} [v_n + d + a \log(1 - v_n/a)], \quad n = 0, 1, 2, \dots \quad (3.24)$$

となる。この式に基づく数値解析の結果を求めて、ドラム缶に亀裂が入るか否かを明らかにせよ。

(出典：M. ブラウン著「微分方程式（上）その数学と応用」, Springer)

第4章

曲線の補間

問題

未来大付近の道路を自動運転車両を走らせる計画を立てた。自動運転車両を走らせるためには道路上を示す位置データ群が密（例えば、1 m 間隔）に必要である。このとき、全ての点を手作業で計測することは困難であり、またバグの原因にもなる。数点の計測結果から、他の点を自動的に生成する方法を考えよ。



計測・実験データ等から、その点を通る関数を推定する方法について学ぶ

演習 3.2.1 放射性廃棄物の海洋投棄は安全か？

アメリカ原子力委員会は、数年にわたって濃縮した放射性廃棄物をドラム缶に密封し、推進 300 ft (約 100m) の地点に海洋投棄していた。この処置に危機感を抱いた生態学者や科学者が問い合わせたところ、原子力委員会は放射能漏れを起こすことは決してないと保証した。ドラム缶の大規模な試験の結果も、原子力委員会の主張を裏打ちするものとなった。しかし、技術者の中から、海底に落ちた時の衝撃から、ドラム缶に亀裂が生じるのではないかとの質問があった。原子力委員会の回答は「ありえない」というものだった。技術者たちは「実験してみればわかる」とし、数多くの実験を行った後、秒速 40ft を超える速度で海底に衝突するとドラム缶に亀裂が生じることがわかった。ここでの問題は、海底に衝突する時点のドラム缶の速度を計算し、秒速 40ft を超えるか否かを明らかにすることである。

海中を落下していくドラム缶の速度は、ニュートン力学から微分方程式を立てることによってモデル化できる。

(微分方程式から、速度を計算するための式の導出過程は、ここでは省略する)

モデル化を通じて、ドラム缶の衝突速度は、次の方程式を満たすことがわかった。

$$g(v) = v + \frac{300cg}{W} + \frac{W-B}{c} \log \left[\frac{W-B-cv}{W-B} \right] = 0 \quad (3.21)$$

ここで

c	0.08	正の定数
g	32.2	重力加速度 (ft/s^2)
W	527.436	ドラム缶が下に引っ張られる重さ
B	470.327	ドラム缶位働く浮力

式 (3.21)において、 $a = (W - B)/c$, $d = 300cg/W$ とすると以下のように単純化できる。

$$g(v) = v + d + a \log(1 - (v/a)) = 0 \quad (3.22)$$

上式の問題に対して、ニュートン法を用いると

$$v_{n+1} = v_n - \frac{g(v_n)}{g'(v_n)} = v_n - \frac{v_n + d + a \log(1 - (v_n/a))}{-\frac{v_n/a}{1-v_n/a}} \quad (3.23)$$

$$= v_n + \frac{a - v_n}{v_n} [v_n + d + a \log(1 - v_n/a)], \quad n = 0, 1, 2, \dots \quad (3.24)$$

となる。この式に基づく数値解析の結果を求めて、ドラム缶に亀裂が入るか否かを明らかにせよ。

(出典：M. ブラウン著「微分方程式（上）その数学と応用」, Springer)

第4章

曲線の補間

問題

未来大付近の道路を自動運転車両を走らせる計画を立てた。自動運転車両を走らせるためには道路上を示す位置データ群が密（例えば、1 m 間隔）に必要である。このとき、全ての点を手作業で計測することは困難であり、またバグの原因にもなる。数点の計測結果から、他の点を自動的に生成する方法を考えよ。



計測・実験データ等から、その点を通る関数を推定する方法について学ぶ

4.1 ラグランジュ補間

4.1.1 考え方

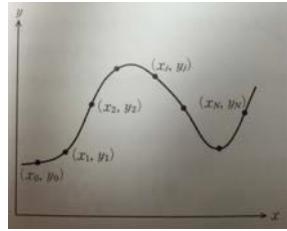
xy 平面内に $N + 1$ 個の点 $(x_0, y_0), (x_1, y_1), \dots, (x_N, y_N)$ が与えられてたとする。これらの点全てを通る曲線を x のたかだか N 次の多項式

$$p_n(x) = \sum_{j=0}^N a_j x^j = \quad (4.1)$$

を用いて、曲線 $y = p_n(x)$ の形で生成することを考える。 $p_N(x)$ は与えられた $N + 1$ 個の点を通らなければならないので、

$$(4.2)$$

を満たさなければならぬ



4.1.2 N=1 のケース

2 点 $(x_0, y_0), (x_1, y_1)$ が与えられたケースを考える。

求める曲線（直線）は

$$p_1(x) = a_0 + a_1 x \quad (4.3)$$

であり、また a_0, a_1 は

$$\begin{cases} a_0 + a_1 x_0 = y_0 \\ a_0 + a_1 x_1 = y_1 \end{cases} \quad (4.4)$$

を解いて

$$\left\{ \begin{array}{l} \\ \end{array} \right. \quad (4.5)$$

ここから

$$p_1(x) = \frac{(x_1 y_0 - x_0 y_1)}{(x_1 - x_0)} + \frac{(y_1 - y_0)}{(x_1 - x_0)} x \quad (4.6)$$

式 (4.6) は「正しい」が、求めるのも煩雑であり、与えられた 2 点を通るかどうか、すぐにわかりにくい。そこで、求める曲線（直線）を次の形にしたものとする。

(4.7)

変数 x が分かれているが、1 次多項式であり、2 点を通るための条件から、未定係数 a, b をすぐに決めることができる。

2 点 $(x_0, y_0), (x_1, y_1)$ を代入すれば、

$$y_0 = a(x_0 - x_1) \quad (4.8)$$

$$y_1 = b(x_1 - x_0) \quad (4.9)$$

上記は、それぞれ独立なので、簡単に a, b を求められる。

$$a = \dots, \quad b = \dots \quad (4.10)$$

したがって、求める直線の式は、

$$y = \dots \quad (4.11)$$

4.1.3 N=2 のケース

3 点 $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ が与えられたケース

2 次のラグランジュ補間多項式を次の様にする。

$$y = \dots \quad (4.12)$$

規則性に注目すれば、各項は $(x - x_0)(x - x_1)(x - x_2)$ から、

順番に $(x - x_0), (x - x_1), (x - x_2)$ を取り除いたものになっている。

3 点を代入すれば

$$y_0 = a(x_0 - x_1)(x_0 - x_2) \quad (4.13)$$

$$y_1 = b(x_1 - x_0)(x_1 - x_2) \quad (4.14)$$

$$y_2 = c(x_2 - x_0)(x_2 - x_1) \quad (4.15)$$

となり、容易に a, b, c を求められる。

$$a = \frac{y_0}{(x_0 - x_1)(x_0 - x_2)} \quad (4.16)$$

$$b = \frac{y_1}{(x_1 - x_0)(x_1 - x_2)} \quad (4.17)$$

$$c = \frac{y_2}{(x_2 - x_0)(x_2 - x_1)} \quad (4.18)$$

したがって、求める式は下記となる。

$$p_2(x) = \frac{(x - \underline{})(x - \underline{})}{(x_0 - \underline{})(x_0 - \underline{})} + \frac{(x - \underline{})(x - \underline{})}{(x_1 - \underline{})(x_1 - \underline{})} + \frac{(x - \underline{})(x - \underline{})}{(x_2 - \underline{})(x_2 - \underline{})} \quad (4.19)$$

4.1.4 一般の N+1 点の場合：ラグランジュの多項式補間

$$p_N(x) = \sum_{j=0}^N \underline{} \quad (4.20)$$

ここで

$$l_j(x) = \frac{(x - x_0)(x - x_1) \dots}{(x_j - x_0)(x_j - x_1) \dots} \frac{(x - x_N)}{(x_j - x_N)} \quad (4.21)$$

分子には $(x - x_j)$ の項、分母は $(x_j - x_i)$ の項が抜けていていることに注意

$l_j(x)$ は N 次多項式で、かつ下記を満たす

$$l_j(\underline{}) = \begin{cases} \underline{} & (i = j) \\ 0 & (i \neq j) \end{cases} \quad (4.22)$$

よって次が成り立つ

$$p_N(x_j) = y_j \quad (4.23)$$

4.1.5 scipy の interpolate の lagrange メソッドを用いた数値計算

- 3 点の標本点 $(x_1, y_1) = (-2, 8), (x_2, y_2) = (2, 4), (x_3, y_3) = (4, 14)$ を通るラグランジュ補間関数

$$y = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 \quad (4.24)$$

- プログラム lagrange1.py を実行して動作を確認すること
- 3 点の標本点は, $y = x^2 - x + 2$ 上の点から作られている（すなわち, 真の関数）.
 $x = 0, x = 1$ のそれぞれにおける y の真の値と補間値が一致しているか確認せよ
- プログラムの中身を理解すること

===== lagrange1.py =====

```

from scipy.interpolate import lagrange
import numpy as np
import matplotlib.pyplot as plt

## メイン

x=[-2, 2, 4]
y=[8, 4, 14]
f_Lag=lagrange(x,y) #scipy.interpolate.lagrange によるラグランジュ補間実行
##

def yy(x):
    return x*x-x+2 # 例題の関数 y = x^2-x+2

#for plot
xnew =np.linspace(-5,5,num=51) # [-5,5] の範囲を 51 等分して xnew に格納

# 正解データの生成
#y_lis_exact=[]
#for j in xnew:
#    y_lis_exact.append(yy(j))

plt.plot(x, y, 'o', xnew, f_Lag(xnew), '--') # 生データを"o"で、 ラグランジュ補間したものを線('--)で描く。
#plt.plot(xnew,y_lis_exact, color='Black',label='Exact') #正解データのプロット
plt.legend(['Raw data','Lagrange'], loc='best') # legend の指定
plt.xlim([-6, 6]) # x 軸のプロット範囲
plt.ylim([0, 16]) # y 軸のプロット範囲
plt.show()

```

参考 : https://qiita.com/sci_Haru/items/4b220d3a67c730ff08dd

演習課題 レポート 3

- (1) レポート提出用ファイルに従って、下記を示せ
- (2) 放射性廃棄物の海洋投棄に関する数値解を求め、その計算結果のスクリーンショットとともに、答えを示し、安全か否か明らかにせよ。
- (3) プログラム lagrange1.py を実行して、描画されたグラフのスクリーンショットを示せ。

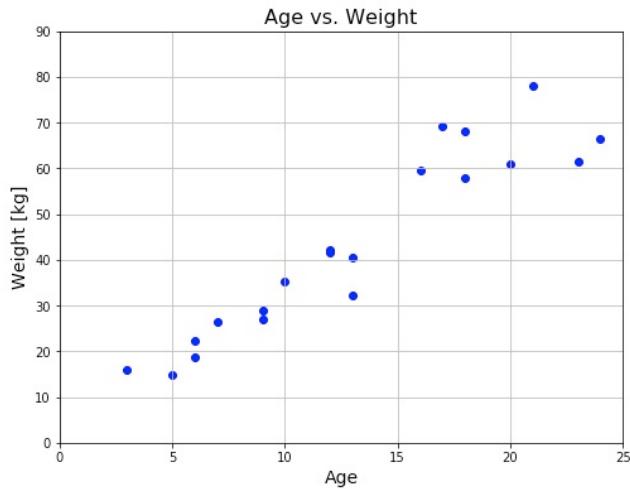
4.3 最小2乗法

問題

図に示した年齢と体重の計測結果から、

- 9歳
- 15歳
- あなたの年齢

のそれぞれの体重は、どれくらいになるのか答えよ。

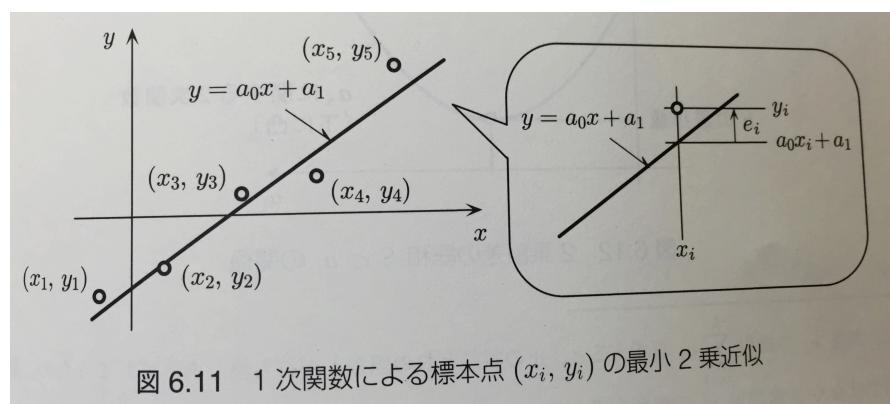


k 個の標本点 $(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)$ が与えられたとき、これらがなるべく直線

(4.58)

と一致するように係数 a_0, a_1 を決定することを考える。

すべての標本点を通るようにこの係数を設定することはできないので
その
を定義し、
を考える。



推定誤差

$$e_i = \quad (4.59)$$

2乗誤差の総和

$$S = \sum_{i=1}^k \{y_i - (a_0 x_i + a_1)\}^2 \quad (4.60)$$

$$= \left(\sum_{i=1}^k x_i^2 \right) - 2 \left\{ \sum_{i=1}^k x_i (y_i - a_1) \right\} + \sum_{i=1}^k (y_i - a_1)^2 \quad (4.61)$$

$$= \left(\sum_{i=1}^k 1 \right) - 2 \left\{ \sum_{i=1}^k x_i (y_i - a_0 x_i) \right\} + \sum_{i=1}^k (y_i - a_0 x_i)^2 \quad (4.62)$$

S の最小化 (a_0, a_1 の最適値の条件) 式 (4.61), 式 (4.62) より S を最小化する条件式は

下記を満足すればよい

$$\begin{cases} = 2 \left(\sum_{i=1}^k x_i^2 \right) a_0 - 2 \left\{ \sum_{i=1}^k x_i (y_i - a_1) \right\} = 0 \\ = 2 \left(\sum_{i=1}^k 1 \right) a_1 - 2 \left\{ \sum_{i=1}^k (y_i - a_0 x_i) \right\} = 0 \end{cases} \quad (4.63)$$

連立 1 次方程式

$$\begin{cases} \left(\sum_{i=1}^k x_i^2 \right) a_0 + \left(\sum_{i=1}^k x_i \right) a_1 = \sum_{i=1}^k x_i y_i \\ \left(\sum_{i=1}^k x_i \right) a_0 + \left(\sum_{i=1}^k 1 \right) a_1 = \sum_{i=1}^k y_i \end{cases} \quad (4.64)$$

$$\Leftrightarrow \begin{pmatrix} \sum_{i=1}^k x_i^2 & \sum_{i=1}^k x_i \\ \sum_{i=1}^k x_i & \sum_{i=1}^k 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^k x_i y_i \\ \sum_{i=1}^k y_i \end{pmatrix} \quad (4.65)$$

$$\Leftrightarrow \quad (4.66)$$

$$(4.67)$$

$$(4.68)$$

■最小2乗法

k 個の標本点 $x_i, y_i \quad (i = 1, 2, \dots, k)$ が与えられたとき、2乗誤差の総和

$$S = \sum_{i=1}^k e_i^2, \quad e_i = y_i - \quad (4.69)$$

が最小となる n 次関数

$$(4.70)$$

の係数 a_0, a_1, \dots, a_n は、連立1次方程式

$$\begin{pmatrix} \sum_{i=1}^k x_i^{2n} & \dots & \sum_{i=1}^k x_i^{n+1} & \sum_{i=1}^k x_i^n \\ \vdots & \ddots & \vdots & \vdots \\ \sum_{i=1}^k x_i^{n+1} & \dots & \sum_{i=1}^k x_i^2 & \sum_{i=1}^k x_i^1 \\ \sum_{i=1}^k x_i^n & \dots & \sum_{i=1}^k x_i^1 & \sum_{i=1}^k x_i^0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^k x_i^n y_i \\ \vdots \\ \sum_{i=1}^k x_i^1 y_i \\ \sum_{i=1}^k x_i^0 y_i \end{pmatrix} \quad (4.71)$$

$$(4.72)$$

を解くことにより求まる

scipy の optimize を使って最小二乗法による補間を行う。

http://www2.kaiyodai.ac.jp/~kentaro/materials/new_HP/python/15fit_data3.html

html

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

def f(x):
    return 5.*x + 10.

#Pusedo measured data by random number
xdata = np.linspace(-10, 10, num=101)
np.random.seed(3456)
ydata = f(xdata) + 5.*np.random.randn(xdata.size)

#Least squares method with scipy.optimize
def fit_func(parameter,x,y):
    a = parameter[0]
    b = parameter[1]
    residual = y-(a*x+b)
    return residual

parameter0 = [0.,0.]
result = optimize.leastsq(fit_func,parameter0,args=(xdata,ydata))
print(result)
a_fit=result[0][0]
b_fit=result[0][1]
print(a_fit,b_fit)

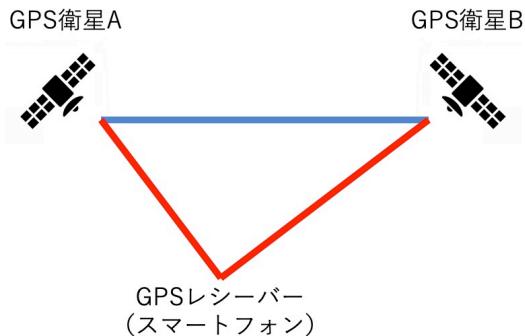
#PLOT
plt.figure(figsize=(8,5))
plt.plot(xdata,ydata,'bo', label='Exp.')
plt.plot(xdata,a_fit*xdata+b_fit,'k-', label='fitted line', linewidth=10, alpha=0.3)
plt.plot(xdata,f(xdata),'r-', label='mother line', linewidth=3)
plt.xlabel('Time')
plt.ylabel('Verocity')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

第5章

連立1次方程式

問題

今あなたは、街中で道に迷ってしまった。自分がいる場所を確認するために、スマートフォンのアプリから自分のいる場所を地図上に表示させようとするだろう。一般的に、GPS レシーバーの位置は、衛星軌道上を移動している複数の GPS 衛星から送られてくる情報（衛星位置と時間）から計算される。このとき、あなたの居る場所を特定するための計算アルゴリズムはどのようなものか答えよ。



- 受信機の位置 $P = (x_p, y_p, z_p)$
- 各 GPS 衛星の位置は, $(x_1, y_1, z_1), \dots, (x_n, y_n, z_n)$
- 各 GPS 衛星と受信との距離は, r_1, \dots, r_n

上記より、受信機の位置 P は、GPS 衛星の位置を中心とし、半径 r_i の球面状にあるといえる。よって次の連立方程式が成り立つ。

$$(x_i - x_p)^2 + (y_i - y_p)^2 + (z_i - z_p)^2 = (r_i + C\Delta t)^2 \quad (5.1)$$

ここで、 Δt は受信機の時計の誤差を表す。この式における変数は、 $x_p, y_p, z_p, \Delta t$ の 4 つであり、この解を求めるためには、4 つ以上の式からなる連立方程式を解く必要がある。

5.1 ガウスの消去法

行基本変形

(1) 1つの行を k 倍する。 $(k \neq 0)$

(2) 1つの行を k 倍して、他の行に加える。

(3) 2つの行を入れかえる。

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ -x_1 + 3x_2 - 5x_3 = -20 \\ x_1 + 2x_2 + 3x_3 = 4 \end{array} \right. \quad (5.2)$$

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ -x_1 + 3x_2 - 5x_3 = -20 \\ x_1 + 2x_2 + 3x_3 = 4 \end{array} \right. \quad (5.3)$$

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ -x_1 + 3x_2 - 5x_3 = -20 \\ x_1 + 2x_2 + 3x_3 = 4 \end{array} \right. \quad (5.4)$$

$$\rightarrow \begin{pmatrix} 2 & -4 & 6 \\ -1 & 3 & -5 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ -20 \\ 4 \end{pmatrix} \quad (5.5)$$

$$\rightarrow \text{拡大係数行列} \left(\begin{array}{ccc|c} 2 & -4 & 6 & 24 \\ -1 & 3 & -5 & -20 \\ 1 & 2 & 3 & 4 \end{array} \right) \quad (5.6)$$

$$\rightarrow \text{前進消去} \left(\begin{array}{ccc|c} 2 & -4 & 6 & 24 \\ 0 & 1 & -2 & -8 \\ 0 & 4 & 0 & -8 \end{array} \right) \rightarrow \left(\begin{array}{ccc|c} 2 & -4 & 6 & 24 \\ 0 & 1 & -2 & -8 \\ 0 & 0 & 8 & 24 \end{array} \right) \quad (5.7)$$

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ x_2 - 2x_3 = -8 \\ 8x_3 = 24 \end{array} \right. \quad (5.8)$$

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ x_2 - 2x_3 = -8 \\ 8x_3 = 24 \end{array} \right. \quad (5.9)$$

$$\left\{ \begin{array}{l} 2x_1 - 4x_2 + 6x_3 = 24 \\ x_2 - 2x_3 = -8 \\ 8x_3 = 24 \end{array} \right. \quad (5.10)$$

$$\rightarrow \text{後退代入 } x_3 = \frac{24}{8} = 3, \quad x_2 = \frac{-8 - (-2x_3)}{1} = -2, \quad x_1 = \frac{24 - (-4x_2 + 6x_3)}{2} = -1 \quad (5.11)$$

5.1.1 3変数連立一次方程式

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{array} \right. \quad (5.12)$$

$$(5.13)$$

$$(5.14)$$

(I) 前進消去 $a_{ij}^{(1)} = a_{ij}, b_i^{(1)}$ として

$$\left(\begin{array}{ccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & b_2^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & b_3^{(1)} \end{array} \right) \quad (5.15)$$

$$\Rightarrow \left(\begin{array}{ccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & b_2^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & b_3^{(2)} \end{array} \right) \quad \begin{array}{l} \textcircled{1} \\ \textcircled{2} - \textcircled{1} \times (a_{21}^{(1)} / a_{11}^{(1)}) \\ \textcircled{3} - \textcircled{1} \times (a_{31}^{(1)} / a_{11}^{(1)}) \end{array} \quad (5.16)$$

$$\Rightarrow \left(\begin{array}{ccc|c} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & b_2^{(2)} \\ 0 & 0 & a_{33}^{(3)} & b_3^{(3)} \end{array} \right) \quad \begin{array}{l} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} - \textcircled{2} \times (a_{32}^{(2)} / a_{22}^{(2)}) \end{array} \quad (5.17)$$

(II) 後退代入

$$x_3 = \frac{b_3^{(3)}}{a_{33}^{(3)}} \quad (5.18)$$

$$x_2 = \frac{b_2^{(2)} - a_{23}^{(2)}x_3}{a_{22}^{(2)}} \quad (5.19)$$

$$x_1 = \frac{b_1^{(1)} - (a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3)}{a_{11}^{(1)}} \quad (5.20)$$

5.1.2 ガウスの消去法のアルゴリズム $a_{kk}^{(k)} \neq 0$ の場合

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad (5.21)$$

$$\rightarrow \mathbf{Ax} = \mathbf{b} \quad \mathbf{A} : \text{係数行列} \quad (5.22)$$

5.1.3 拡大係数行列

$$(\mathbf{A}, \mathbf{b}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix} \quad (5.23)$$

ステップ 1 $i, j = 1, 2, \dots, n$ に対し, $a_{ij}^{(1)} = a_{ij}$, $b_j^{(1)} = b_j$ とする

ステップ 2 (前進消去) $k = 1, 2, \dots, n-1$ の順に以下の計算を行う

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{kj}^{(k)} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad (i, j = k+1, k+2, \dots, n) \quad (5.24)$$

$$b_i^{(k+1)} = b_i^{(k)} - b_k^{(k)} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \quad (i = k+1, k+2, \dots, n) \quad (5.25)$$

これらの操作により, 係数は以下のように変化する.

$$\left(\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \dots & a_{2n}^{(1)} & b_2^{(1)} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} & b_n^{(1)} \end{array} \right) \quad (5.26)$$

$$\Rightarrow \dots \quad (5.27)$$

$$\Rightarrow \left(\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} & b_2^{(2)} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & \dots & 0 & a_{nn}^{(n)} & b_n^{(n)} \end{array} \right) \quad (5.28)$$

ステップ3 (後退代入) x_n を

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \quad (5.29)$$

により求めた後, $i = n - 1, n - 2, \dots, 1$ の順に下記を求める.

$$x_i = \frac{1}{a_{ii}^{(i)}} \left(b_i^{(i)} - \sum_{k=i+1}^n a_{ik}^{(i)} x_k \right) \quad (5.30)$$

5.1.4 ピボット操作

前進消去における $a_{kk}^{(k)}$: ピボットとよぶ

- (1) 前述のアルゴリズムは $a_{kk}^{(k)} \neq 0$ を仮定, $a_{kk}^{(k)} = 0$ のとき, 前進消去ができなくなる.
- (2) $a_{lk}^{(k)}$ が微小である場合, 丸め誤差による誤答やオーバーフローを起こす

ピボット操作により, 行を入れ替えて回避を行う

行交換によるピボット操作

- (1) $a_{ik}^{(k)}$ ($i = k, k + 1, \dots, n$) の中で絶対値が最大のものをピボットに選ぶ
- (2) $|a_{lk}^{(k)}|$ が最大ならば, l 番目と k 番目の行 (方程式) を交換する

5.2 LU 分解

ガウスの消去法は、計算の効率という点から良い方法とは言えない。ここでは、より効率の高い LU 分解について示す。

- L : 下三角行列 式 (5.32)
- U : 上三角行列 式 (5.33)
- $LU = A$ となるように、 L, U を求めるのが、LU 分解
- $Ax = b$ を $LUx = b$ として連立方程式を解く。

$$A = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix} \quad (5.31)$$

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{10} & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & 0 \\ l_{n-1,0} & l_{n-1,1} & \cdots & 1 \end{pmatrix} \quad (5.32)$$

$$U = \begin{pmatrix} u_{00} & u_{01} & \cdots & u_{0,n-1} \\ 0 & u_{11} & \cdots & u_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & u_{n-1,n-1} \end{pmatrix} \quad (5.33)$$

$$a_{ij} = \begin{cases} l_{i0}u_{0j} + l_{i1}u_{1j} + \cdots + l_{i,i-1}u_{i-1,j} + u_{ij} & (i \leq j) \\ l_{i0}u_{0j} + l_{i1}u_{1j} + \cdots + l_{i,i-1}u_{i-1,j} + l_{ij}u_{jj} & (j < i) \end{cases} \quad (5.34)$$

5.2.1 概要

LU 分解の大まかな手順は以下の通りである

- (1) $A\mathbf{x} = \mathbf{y}$ と問題を設定する
- (2) 行列 A を下三角行列 L と上三角行列 U に分解 : $A = LU$ とすると, $LU\mathbf{x} = \mathbf{y}$
- (3) 未知ベクトル \mathbf{z} を用意し, $\mathbf{z} = U\mathbf{x}$ とする.
 $L\mathbf{z} = \mathbf{y}$ として, \mathbf{z} について解く (前進代入)
- (4) $U\mathbf{x} = \mathbf{z}$ を \mathbf{x} について解く (後退代入)

5.2.2 数値例

例題 5.2.1 LU 分解を用いて, 次の連立一次方程式を解け

$$\begin{cases} 2x_1 + 2x_2 + 6x_3 = 24 \\ 3x_1 + 5x_2 + 13x_3 = 52 \\ 5x_1 + 8x_2 + 24x_3 = 93 \end{cases} \quad (5.35)$$

式 (5.35) より, 行列 A を次のように定める.

$$A = \begin{pmatrix} 2 & 2 & 6 \\ 3 & 5 & 13 \\ 5 & 8 & 24 \end{pmatrix} \quad (5.36)$$

計算状態の表現として $\langle M_t, A_t \rangle$ を用意する. ここで, t は t 回目の行列操作とする.

$$\langle M_0, A_0 \rangle = \langle I, A \rangle \quad (5.37)$$

すなわち, 以下を用意する.

$$\left\langle I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A = \begin{pmatrix} 2 & 2 & 6 \\ 3 & 5 & 13 \\ 5 & 8 & 24 \end{pmatrix} \right\rangle \quad (5.38)$$

ここから, A が上三角行列になるように, 行列変形を行っていく. この際, 行を変形するためを使った乗数を M に記憶させる.

$$\text{第2行} \leftarrow \text{第2行} - \text{第1行} \times (3/2) \quad (5.39)$$

$$\Rightarrow \left\langle M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A_1 = \begin{pmatrix} 2 & 2 & 6 \\ 0 & 2 & 4 \\ 5 & 8 & 24 \end{pmatrix} \right\rangle \quad (5.40)$$

$$\text{第3行} \leftarrow \text{第3行} - \text{第1行} \times (5/2) \quad (5.41)$$

$$\Rightarrow \left\langle M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 5/2 & 0 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} 2 & 2 & 6 \\ 0 & 2 & 4 \\ 0 & 3 & 9 \end{pmatrix} \right\rangle \quad (5.42)$$

$$\text{第3行} \leftarrow \text{第3行} - \text{第2行} \times (3/2) \quad (5.43)$$

$$\Rightarrow \left\langle M_3 = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 5/2 & 3/2 & 1 \end{pmatrix} = L, A_3 = \begin{pmatrix} 2 & 2 & 6 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{pmatrix} = U \right\rangle \quad (5.44)$$

ここで、得られた L, U について、以下を確かめよ。

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 3/2 & 1 & 0 \\ 5/2 & 3/2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 2 & 6 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 6 \\ 3 & 5 & 13 \\ 5 & 8 & 24 \end{pmatrix} = A \quad (5.45)$$

方程式 $Ax = b$ は LU 分解から

$$LUx = b \quad (5.46)$$

と書くことができる。ここで、 $Ux = y$ とおくと、次の2つの連立一次方程式が得られる。

$$Ly = b \quad (5.47)$$

$$Ux = y \quad (5.48)$$

よって、まず $Ly = b$ の解 y を求め、次に $Ux = y$ を解けば、 x が求まる。

$$\begin{cases} y_1 &= 24 \\ \frac{3}{2}y_1 + y_2 &= 52 \end{cases} \quad (5.49)$$

$$\begin{cases} \frac{5}{2}y_1 + \frac{3}{2}y_2 + y_3 &= 93 \end{cases} \quad (5.50)$$

$$\begin{cases} 2x_1 + 2x_2 + 6x_3 &= y_1 \\ 2x_2 + 4x_3 &= y_2 \\ 3x_3 &= y_3 \end{cases} \quad (5.51)$$

上記より $y_1 = 24, y_2 = 16, y_3 = 9$

$$\begin{cases} 2x_1 + 2x_2 + 6x_3 &= y_1 \\ 2x_2 + 4x_3 &= y_2 \\ 3x_3 &= y_3 \end{cases} \quad (5.52)$$

上記より $x_1 = 1, x_2 = 2, x_3 = 3$

例題 5.2.2 LU 分解を用いて、次の 4 元連立 1 次方程式を解け

$$\left\{ \begin{array}{l} 4x_0 + x_1 - 3x_2 + 5x_4 = 18 \\ 2x_0 + 3x_1 + x_2 + 2x_4 = 21 \\ 5x_0 - 4x_1 - 2x_2 + x_4 = 5 \\ -2x_0 + 2x_1 + 8x_2 + 2x_4 = 16 \end{array} \right. \quad (5.53)$$

(1) A を LU に分解

(1) U の第 0 行を求める。式 (121) より $a_{0j} = u_{0j}$ 、よって

$$u_{00} = 4, u_{01} = 1, u_{02} = -3, u_{03} = 5$$

(2) L の第 0 列を求める。式 (121) より $j = 0$ とすると $a_{i0} = l_{i0}u_{00}$ 、よって $l_{i0} = \frac{a_{i0}}{u_{00}}$ から A の第 0 列を $u_{00} = 4$ で割る

$$l_{00} = 1, l_{10} = 0.5, l_{20} = 1.25, l_{30} = -0.5$$

(3) U の第 1 行

$$u_{1j} = a_{1j} - (l_{10}u_{0j}) \quad \text{より} \quad (5.54)$$

$$u_{11} = 3 - (0.5 \times 1) = 2.5 \quad (5.55)$$

$$u_{12} = 1 - (0.5 \times (-3)) = 2.5 \quad (5.56)$$

$$u_{13} = 2 - (0.5 \times 5) = -0.5 \quad (5.57)$$

(4) L の第 1 列

$$a_{i1} = l_{i0}u_{01} + l_{i1}u_{11} \quad \text{より} \quad (5.58)$$

$$l_{i1} = \frac{1}{u_{1,1}}(a_{i1} - l_{i0}u_{01}) \quad \text{となるので} \quad (5.59)$$

$$l_{21} = (-4 - 1.25 \times 1)/2.5 = -5.25/2.5 = -2.1 \quad (5.60)$$

$$l_{31} = (2 - (-0.5) \times 1)/2.5 = -2.5/2.5 = 1 \quad (5.61)$$

(5) U の第 2 行

$$u_{2j} = a_{2j} - (l_{20}u_{0j} + l_{21}u_{1j}) \quad \text{より} \quad (5.62)$$

$$u_{22} = -2 - (1.25 \times (-3) + (-2.1) \times 2.5) = 7.0 \quad (5.63)$$

$$u_{23} = 1 - (1.25 \times 5 + (-2.1) \times (-0.5)) = -6.3 \quad (5.64)$$

(6) L の第 2 列

$$a_{i2} = l_{i0}u_{02} + l_{i1}u_{12} + l_{i2}u_{2,2} \quad \text{より} \quad (5.65)$$

$$l_{i2} = \frac{1}{u_{2,2}}(a_{i2} - (l_{i0}u_{02} + l_{i1}u_{12})) \quad \text{となるので} \quad (5.66)$$

$$l_{32} = (8 - (-0.5 \times (-3) + 1.0 \times 2.5))/7 \cong 0.571 \quad (5.67)$$

(7) U の第 3 行

$$u_{3j} = a_{3j} - (l_{30}u_{0j} + l_{31}u_{1j} + l_{32}u_{2j}) \quad \text{より} \quad (5.68)$$

$$u_{33} = 2 - (-0.5 \times 5 + 1.0 \times (-0.5) + 0.571 \times (-6.3)) = 8.6 \quad (5.69)$$

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & -3 & 5 \\ 2 & 3 & 1 & 2 \\ 5 & -4 & -2 & 1 \\ -2 & 2 & 8 & 2 \end{pmatrix} \quad (5.70)$$

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 1 & 0 & 0 \\ 1.25 & -2.1 & 1 & 0 \\ -0.5 & 1.0 & 0.571 & 1 \end{pmatrix} \quad (5.71)$$

$$\mathbf{U} = \begin{pmatrix} 4 & 1 & -3 & 5 \\ 0 & 2.5 & 2.5 & -0.5 \\ 0 & 0 & 7 & -6.3 \\ 0 & 0 & 0 & 8.6 \end{pmatrix} \quad (5.72)$$

$Lz = y$ より z を求める (前進代入)

$$\mathbf{L} = \begin{cases} z_0 & = 18 \\ 0.5z_0 & + z_1 = 21 \\ 1.25z_0 & - 2.1z_1 + z_2 = 5 \\ -0.5z_0 & + z_1 + 0.571z_2 + z_3 = 16 \end{cases} \quad (5.73)$$

よって

$$z_0 = 18 \quad (5.74)$$

$$z_1 = 21 - 0.5 \times 18 = 12 \quad (5.75)$$

$$z_2 = 5 - 1.25 \times 18 + 2.1 \times 12 = 7.7 \quad (5.76)$$

$$z_3 = 16 + 0.5 \times 18 - 12 - 0.571 \times 7.7 = 8.6 \quad (5.77)$$

$Ux = z$ より x を求める (後退代入)

$$\mathbf{U} = \begin{cases} 4x_0 & x_1 & -3x_2 & +5x_3 & = 18 \\ & 2.5x_1 & +2.5x_2 & -0.5x_3 & = 12 \\ & & 7x_2 & -6.3x_3 & = 7.7 \\ & & & 8.6x_3 & = 8.6 \end{cases} \quad (5.78)$$

よって

$$x_3 = 8.6/8.6 = 1.0 \quad (5.79)$$

$$x_2 = (7.7 + 6.3 \times 1.0)/7 \quad (5.80)$$

$$x_1 = (12 - 2.5 \times 2.0 + 0.5 \times 1.0)/2.5 = 3.0 \quad (5.81)$$

$$x_0 = (18 - 3.0 + 3 \times 2.0 - 5 \times 1.0)/4.0 = 4.0 \quad (5.82)$$

5.2.3 アルゴリズム

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \\ &= \mathbf{L}\mathbf{U} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \dots & 0 \\ l_{n,1} & l_{n,2} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1,n} \\ 0 & u_{22} & \dots & u_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & u_{n,n} \end{pmatrix} \end{aligned} \quad (5.83)$$

- (1) $Ax = y$ の問題設定 (n, a_{ij}, y_i)
(2) $i = 1, 2, \dots, n$ の順に以下繰り返す
(2.1) $j = 1, 2, \dots, i - 1$ の順に以下繰り返す

$$l_{ij} = \frac{1}{u_{ji}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \quad (5.84)$$

- (2.2) $j = i, i + 1, \dots, n$ の順に以下繰り返す

$$u_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \quad (5.85)$$

- (3) 前進代入

$$z_1 = y_1 \quad (5.86)$$

- $i = 2, 3, \dots, n$ の順に以下繰り返す

$$z_i = y_i - \sum_{k=1}^{i-1} l_{ik} z_k \quad (5.87)$$

- (4) 後退代入

$$x_n = z_n / u_{nn} \quad (5.88)$$

- $i = n - 1, n - 2, \dots, 1$ の順に以下繰り返す

$$x_i = \left(z_i - \sum_{k=i+1}^n u_{ik} x_k \right) / u_{ii} \quad (5.89)$$

上記アルゴリズムでは、ガウスの消去法と同じく、ピボット選択が一般的には必要であるが、省略している。

5.2.4 アルゴリズムの評価

LU 分解の乗除算回数 C_1

- U の要素 u_{kj} 一つ求めるために, $l_{k0}u_{0j}$ から $l_{k,k-1}u_{k-1,j}$ までの k 回の乗算
- k 行目のすべての要素を求めるために, $j = k, k+1, \dots, n-1$ の k 回繰り返す
- L の要素 l_{ik} 一つ求めるために, $l_{i0}u_{0k}$ から $l_{i,k-1}u_{k-1,k}$ までの k 回の乗算と u_{kk} での 1 回の除算
- すべての要素を求めるために, $i = k+1, k+2, \dots, n-1$ の $n-k-1$ 回繰り返す
- 以上の計算を $k = 0, 1, \dots, n-1$ について実行

$$C_1 = \sum_{k=1}^{n-1} k(n-k) + \sum_{k=0}^{n-1} (k+1)(n^k - 1) = \frac{1}{2}n^3 - \frac{1}{3}n \quad (5.90)$$

$Lz = y$ より z を求める乗算回数 C_2 各 z_i を求めるために 0 から $i-1$ までの i 回の乗算を行い, $i = 1, 2, \dots, n-1$ まで繰り返すので

$$C_2 = \sum_{i=1}^{n-1} i = \frac{1}{2}n^2 - \frac{1}{2}n \quad (5.91)$$

$Ux = z$ より x を求める乗除算回数 C_3 各 x_i を求めるために $i+1$ から $n-1$ までの $n-i-1$ 回の乗算と 1 回の除算, すなわち, $n-i$ 回の乗除算を行い, これを $i = n-1, n-2, \dots, 0$ まで逆順に繰り返すので

$$C_3 = \sum_{i=0}^{n-1} (n-i) = \frac{1}{2}n^2 - \frac{1}{2}n \quad (5.92)$$

全体の乗除算回数 C_{LU}

$$C_{LU} = C_1 + C_2 + C_3 = \frac{1}{3}n^3 + n^2 - \frac{1}{3}n \quad (5.93)$$

全体の計算オーダーはガウスの消去法 $O(n^3)$ と変わらない. しかしガウスの消去法が前進代入, 後退代入それぞれにも $O(n^3)$ であるのに対し, LU 分解ではそれぞれ $O(n^2)$ ですむ利点がある.

$A x = b$ を満たす x ベクトルを `linalg.solve` を使って求める

https://qiita.com/sci_Haru/items/0b34d730ed8533f921f9

```
import numpy as np

"""
A x = b を満たす x ベクトルを linalg.solve を使って求める
https://qiita.com/sci_Haru/items/0b34d730ed8533f921f9
"""

# 行列 A の生成
a1_lis = [1, 0, 0]
a2_lis = [0, 2, 0]
a3_lis = [0, 0, 4]
A_matrix=np.array([a1_lis, a2_lis, a3_lis])

# b vector
b = np.array([4,5,6]) # 行ベクトルとして b を生成し、その転置行列として"列ベクトル"
# を生成する

x_vec= np.linalg.solve(A_matrix,b) # x ベクトルを計算する。行列の積に@演算子を使っ
# てはいる (参考)

print(x_vec)
```

演習課題 レポート 6

1 最小二乗法課題

擬似データを使った推定

lstsq1.ipynb は、ある関数 $y = ax + b$ に対して、乱数 r_i を付加して

$y_i = ax_i + b + r_i$ から、推定に使う元データ (x_i, y_i) を生成している。

- (1) 最小二乗法のプログラム lstsq1.ipynb を動かして、その結果を示せ。
- (2) プログラム中の元となっている関数” $2.78*x + 1.32$ ”の式を変更してデータを作成し、推定結果と比較し考察せよ。

年齢と体重に関するデータからの推定

lstsq-age-wieght.ipynb は、ある年齢と体重に関する計測データから、最小二乗法により推定値を出すプログラムである。

- (1) このプログラムを動かして、推定した関数を示せ。
- (2) 推定した関数から、9歳、15歳、あなたの年齢、のそれぞれの体重はいくらとなるか推定せよ。

2 連立一次方程式課題

$$\begin{cases} -3x_2 + 2x_3 = 4 \\ x_1 - x_2 - 2x_3 = 5 \end{cases} \quad (5.94)$$

$$\begin{cases} x_1 - x_2 - 2x_3 = 5 \\ 4x_1 + x_2 - 2x_3 = 4 \end{cases} \quad (5.95)$$

$$\begin{cases} x_1 - x_2 - 2x_3 = 5 \\ 4x_1 + x_2 - 2x_3 = 4 \end{cases} \quad (5.96)$$

が与えられたとき、以下の設問に答えよ

- (1) プログラム matrix1.py を実行し、その計算結果を確認せよ。
- (2) 上述の連立方程式を解くように、matrix1.py を変更し、その結果を示せ。

6.4 モンテカルロ積分法

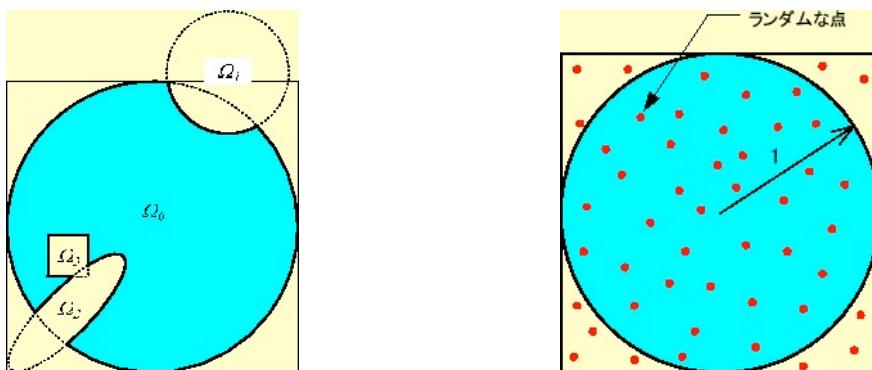
問題

日本の彫刻の傑作、東大寺の金剛力士像（国宝）。修復費用見積のため、この彫像の表面積と体積を求めよ。

<https://nara-jisya.info/南大門/>



より複雑な関数の積分や、その複雑な関数が重なり合った場合などとなると、前節の方法では計算が困難となる。そのような場合に対応した方法として、一風変わった確率的方法論に基づくモンテカルロ積分法を紹介する。



6.4.1 単位円の面積

- 単位円とその円を囲む正方形を考え、正方形内部にランダムに点を打つ:
- 単位円内部にある点の数:
- 正方形面積： S_s 円面積： S_c
- : 点の数が十分に大きくなれば、上記式は等式とみなすことが可能となると期待される

- 正方形面積はわかっているので、求める面積 S_c は

$$S_c \simeq \frac{N_c}{N_s} S_s \quad (6.20)$$

- 精度はサンプル数の平方根に比例：誤差を $1/10$ にしたければ、サンプル数を 100 倍にしなければならない

演習課題 レポート 9

モンテカルロ積分法により関数 $\int_0^1 \frac{4}{1+x^2} dx = \pi$ を計算するプログラム

`simple_Monte_Carlo.py`

を実行し、下記を記してレポートを提出せよ。

- 1) 実行結果を載せること。
- 2) 実行結果の各行が意味することを、述べ考察せよ。

(参考 URL) [Python による科学・技術計算] モンテカルロ積分、数値計算、numpy

https://qiita.com/sci_Haru/items/94a767bed0e512398e2e

```
import numpy as np
from random import random
"""
単純モンテカルロ積分法：試行回数 N を 10 から  $10^7$  まで変化させる
"""
def f(x):
    return 1.0/(1.0+x**2) # 関数の定義

N_calc_list = [10, 10**2, 10**3, 10**4, 10**5, 10**6, 10**7]

for N in N_calc_list:
    count = 0.0
    for i in range(N):
        x = random() # [0,1] までの一様乱数を x に格納
        y = random() # [0,1] までの一様乱数を y に格納
        if y < f(x): #もし"マト"に入ったらそれをカウントする
            count +=1.0
    area = 4*count/N # 積分結果。 π の評価

    print(N, ", ", area, ", ", abs((np.pi-area)/np.pi)) #結果の出力
```

5.3 反復法

これまでに示した、ガウスの消去法や LU 分解などは、原理的にはあらゆる連立方程式を有限回の手続きで必ず解を得ることができる。しかし、メモリの使用量など、現実問題では大規模な問題では、必ずしも良い方法とは言えない場合がある。ここでは、反復計算により、近似解を求める方法である、SOR 法について示す。SOR 法は、以下に示す 2 つの基礎的な解法を基礎とする。

5.3.1 ヤコビ法

$$\begin{cases} 5x_1 + 4x_2 = 13 \\ 2x_1 + 3x_2 = 8 \end{cases} \quad (5.97)$$

上記方程式の解 : $(x_1, x_2) = (1, 2)$

式 (5.97) を次のように変形する。

$$\begin{cases} x_1 = \\ x_2 = \end{cases} \quad (5.98)$$

上記式から漸化式を作る。

$$\begin{cases} x_1 = \frac{1}{5}(13 - 4x_2) = f_1(x_2) \\ x_2 = \frac{1}{3}(8 - 2x_1) = f_2(x_1) \end{cases} \quad (5.99)$$

式 (5.99) に適当な初期値 $(x_1^{(0)}, x_2^{(0)})$ をいれると、 $k = 1, 2, \dots$ に応じて計算を進めることができる。

表 5.1 ヤコビ法の計算結果

k	$x_1^{(k)}$	$x_2^{(k)}$
0	0	0
1	2.60	2.66666
2	0.4666	0.93333
3	1.8533333	2.35555
\vdots	\vdots	\vdots
40	0.99999653	1.99999307
41	1.00000555	2.00000231

解への収束 問題は、必ず収束するのかどうかにある

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{pmatrix} = \quad (5.100)$$

この式において解 $(x_1, x_2) = (1, 2)$ は以下を充たす。

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} = M \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} \frac{13}{5} \\ \frac{8}{3} \end{pmatrix} \quad (5.101)$$

上記 2 式の差を取る

$$(5.102)$$

よって、初期値から次の式を得る

$$\begin{pmatrix} x_1^{(k+1)} - 1 \\ x_2^{(k+1)} - 2 \end{pmatrix} = \begin{pmatrix} x_1^{(0)} - 1 \\ x_2^{(0)} - 2 \end{pmatrix} \quad (5.103)$$

M の固有値、固有ベクトル

M の固有方程式

$$(5.104)$$

固有値 : $\lambda_1 = \dots$, $\lambda_2 = \dots$

固有ベクトル : $e_1 = \dots$, $e_2 = \dots$ よって、式 (5.103) は以下のように表される。

$$\begin{pmatrix} x_1^{(k+1)} - 1 \\ x_2^{(k+1)} - 2 \end{pmatrix} = \quad (5.105)$$

$$(5.106)$$

ここで、

$$\lim_{k \rightarrow \infty} \begin{pmatrix} x_1^{(k)} - 1 \\ x_2^{(k)} - 2 \end{pmatrix} = \quad (5.107)$$

- よって任意の初期解に対して、 $x_1^{(k)}, x_2^{(k)}$ は解に収束
- 収束するためには、行列 M の固有値がすべて

5.3.2 ガウス-ザイデル法

$$\begin{cases} x_1^{(k+1)} = \frac{1}{5}(12 - 4x_2^{(k)}) = f_1(x_2^{(k)}) \\ x_2^{(k+1)} = \frac{1}{3}(8 - 2x_1) = f_2(x_1) \end{cases} \quad (5.108)$$

- 第1式で $x_1^{(k+1)}$ を求めたら、その値を $x_2^{(k+1)}$ を求めるためにすぐ使う
- 初期値 $x_2^{(0)}$ のみ、 $x_1^{(0)}$ は知らない: $x_2^{(0)} \rightarrow x_1^{(1)} \rightarrow x_2^{(1)} \rightarrow x_1^{(2)} \rightarrow x_2^{(2)} \rightarrow \dots$
- 解への収束速度が、ヤコビ法のほぼ2倍

$$\text{ガウス - ザイデル法: } x_1^{(k+1)} = f_1(x_2^{(k)}) = f_1(f_2(x_1^{(k)})) \quad (5.109)$$

$$\text{ヤコビ法: } x_1^{(k+2)} = f_1(x_2^{(k+1)}) = f_1(f_2(x_1^{(k)})) \quad (5.110)$$

$$(5.111)$$

表 5.2 ガウス-ザイデル法の計算結果

k	$x_1^{(k)}$	$x_2^{(k)}$
0	-	0
1	2.60	0.93333
2	1.8533333	1.43111
3	1.4551111	1.69659259
\vdots	\vdots	\vdots
19	1.00001950	1.99998700
20	1.00001040	1.99999307

5.3.3 SOR 法

SOR: successive overrelaxation method, 逐次過緩和法

パラメータ ω (加速係数) を用いて以下のように修正

$$\begin{cases} x_1^{(k+1)} = \frac{1}{5}(12 - 4x_2^{(k)}) \\ x_2^{(k+1)} = \frac{1}{3}(8 - 2x_1^{(k+1)}) \end{cases} \quad (5.112)$$

- ω および初期値 $x_1^{(0)}, x_2^{(0)}$ を定める : $x_1^{(1)} \rightarrow x_2^{(1)} \rightarrow x_1^{(2)} \rightarrow x_2^{(2)} \rightarrow \dots$
- パラメータ ω (加速係数) が 1 のとき, ガウス-ザイデル法に
- 反復計算の変化量 :

$$\begin{cases} x_1^{(k+1)} - x_1^{(k)} = \left\{ f_1(x_2^{(k)}) - x_1^{(k)} \right\} \\ x_2^{(k+1)} - x_2^{(k)} = \left\{ f_2(x_1^{(k+1)}) - x_2^{(k)} \right\} \end{cases} \quad (5.113)$$

ガウス-ザイデル法の計算による変化を, ω を用いて

- 一般に最適な ω は の範囲に存在し, ヤコビ法, ガウス-ザイデル法よりも速く収束

表 5.3 SOR 法 ($\omega = 1.2$) の計算結果

k	$x_1^{(k)}$	$x_2^{(k)}$
0	0	0
1	3.12000000	0.70400000
2	1.82016000	1.60307200
3	1.21701888	1.90577050
4	1.04705655	1.98120066
5	1.00863605	1.99685102
6	1.00129581	1.99959315
7	1.00013141	1.99997624
8	0.99999653	2.00000753
\vdots	\vdots	\vdots

5.3.4 一般の連立方程式における漸化式

係数行列

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1,n} \\ a_{21} & a_{22} & \dots & a_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ a_{N,1} & a_{N,2} & \dots & a_{N,N} \end{pmatrix} \quad (5.114)$$

連立方程式 $\mathbf{Ax} = \mathbf{y}$ として、ある x_i について変形すれば

$$x_i = \frac{1}{a_{i,i}} \left\{ y_i - (a_{i,1}x_1 + a_{i,2}x_2 + \dots + a_{i,N}x_N) \right\} \quad (5.115)$$

$$= \frac{1}{a_{i,i}} \left\{ y_i - \left(\sum_{j=0}^{i-1} a_{i,j}x_j + \sum_{j=i+1}^N a_{i,j}x_j \right) \right\} \quad (5.116)$$

ヤコビ法の漸化式

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left\{ y_i - \left(\sum_{j=0}^{i-1} a_{i,j}x_j + \sum_{j=i+1}^N a_{i,j}x_j^{(k)} \right) \right\} \quad (5.117)$$

ガウス-ザイデル法の漸化式

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left\{ y_i - \left(\sum_{j=0}^{i-1} a_{i,j}x_j + \sum_{j=i+1}^N a_{i,j}x_j^{(k)} \right) \right\} \quad (5.118)$$

SOR 法の漸化式

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left\{ y_i - \left(\sum_{j=0}^{i-1} a_{i,j}x_j^{(k+1)} + \sum_{j=i+1}^N a_{i,j}x_j^{(k)} \right) \right\} \quad (5.119)$$

収束判定条件

- 小さな正数 ε を用いて

$$\frac{1}{N} \sum_{i=1}^N |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon \frac{1}{N} \sum_{i=1}^N |x_i^{(k+1)}| \quad (5.120)$$

「成分で考えて、平均すると、 $x_i^{(k+1)}$ と $x_i^{(k)}$ の差が、 $x_i^{(k+1)}$ の大きさの ε 倍未満に小さくなる」

- 実際の計算では、両辺 N で割ることは不要

$$\sum_{i=1}^N |x_i^{(k+1)} - x_i^{(k)}| < \varepsilon \sum_{i=1}^N |x_i^{(k+1)}| \quad (5.121)$$

```

# http://fornext1119.web.fc2.com/NumericOperation/vol_06/Text/10_02_06.xhtml
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

N = 4

# 1次元配列を表示
def disp_vector(row):
    for col in row:
        print('%14.10f' % col)
    print("")

# ガウス・ザイデル法
def gauss(a, b, x0):
    while True:
        x1 = 0.0
        finish = True
        for i in range(0, N, 1):
            x1 = 0
            for j in range(0, N, 1):
                if (j != i):
                    x1 += a[i][j] * x0[j]

            x1 = (b[i] - x1) / a[i][i]
            if (abs(x1 - x0[i]) > 0.0000000001):
                finish = False
            x0[i] = x1

        if (finish):
            return

    disp_vector(x0)

a = [[ 9.0,  2.0,  1.0,  1.0], [2.0,  8.0, -2.0,  1.0], [-1.0, -2.0,  7.0, -2.0], [1.0, -1.0, -2.0,  6.0]]
b = [20.0, 16.0, 8.0, 17.0]
c = [ 0.0,  0.0,  0.0,  0.0]

# ガウス・ザイデル法
gauss(a,b,c)

print("X")
disp_vector(c)

```

演習課題 レポート 6

1 連立方程式 反復法課題

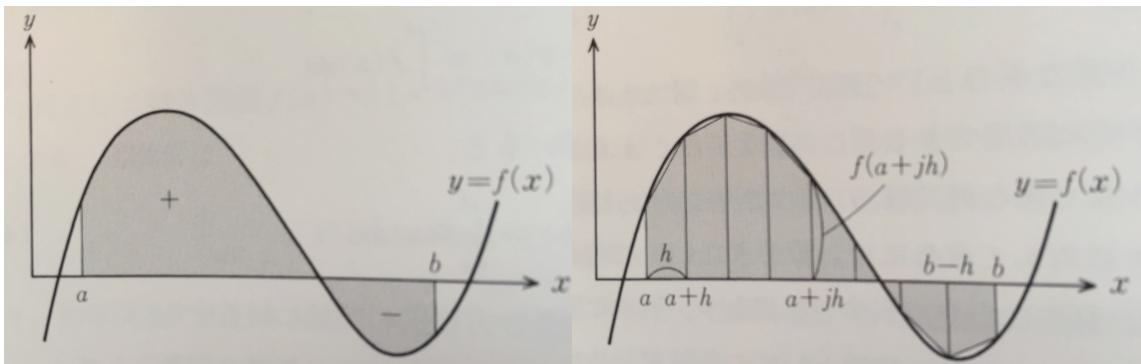
- (1) プログラム gauss_seidel1.jpynb を動かして解いた連立方程式とその解, 反復回数を示せ.
- (2) gauss_seidel.jpynb により, 例題1を解き, その実行結果を示せ.
- (3) gauss_seidel.jpynb に必要な修正を加えて, SOR 法として実行し, その結果とガウス・ザイデル法と比較せよ.
- (4) SOR-Loop.jpynb の実行結果を示せ.
- (5) SOR-Loop.jpynb の実行結果から, パラメータ（加速係数）の変化に伴って答えを求める過程がどのように変化するか考察せよ.

第 6 章

積分

$$\text{定積分} : I = \int_a^b f(x) dx \quad (6.1)$$

6.1 台形則



- (1) $f(x) > 0$ の部分の面積は正, $f(x) < 0$ の部分の面積は負として計算
- (2) $x = a$ から $x = b$ まで N 個の台形が並んでいるとすると
- (3) 台形の横幅 :
- (4) 左端の台形を 0 番目, 右端の台形を $N - 1$ 番目とする
- (5) j 番目の台形の上底: 下底 :
- (6) 各 x 座標の値
- (7) 全部の台形の面積の和: 台形則

$$I \simeq T =$$

=

$$(6.2)$$

- (8) N を大きくして分割を細かくすれば, 計算精度が上がる:

(9) 台形則は関数 $f(x)$ を折れ線関数 $F(x)$ で近似している：1次のラグランジュ補間

$$F(x) = \frac{x - x_{j+1}}{x_j - x_{j+1}} f(x_j) + \frac{x - x_j}{x_{j+1} - x_j} f(x_{j+1}) \quad (x_j \leq x \leq x_{j+1}) \quad (6.3)$$

6.1.1 計算終了条件と計算手順

計算終了条件 分割数 N を増やしながら、求まった答えに関して、 N を変えても先頭から p 術変化しなければ、計算終了とする

計算手順— N を増やす方法 N を変えるごとに、すべての分点を計算するのは効率が悪い

(a) N の増やし方 : $N = 2^n$ ($n = 0, 1, 2, \dots$) で増やしていく

(b) $N = 2^n$ の分点は $N = 2^{n+1}$ に含まれる

(c) $N = 2^n$ における台形則の答えを T_n とする。

このとき台形の幅は

$$\begin{aligned} T_n &= \frac{b-a}{2^n} \left\{ \frac{f(a)}{2} + f\left(a + 1 \cdot \frac{b-a}{2^n}\right) + f\left(a + 2 \cdot \frac{b-a}{2^n}\right) + \cdots + \frac{f(b)}{2} \right\} \\ &= \frac{b-a}{2^n} \left\{ \frac{f(a)}{2} + f\left(a + 2 \cdot \frac{b-a}{2^n}\right) + f\left(a + 4 \cdot \frac{b-a}{2^n}\right) + \cdots + \frac{f(b)}{2} \right\} \end{aligned} \quad (6.4)$$

$$\begin{aligned} T_{n+1} &= \frac{b-a}{2^{n+1}} \left\{ \frac{f(a)}{2} + f\left(a + 1 \cdot \frac{b-a}{2^{n+1}}\right) + f\left(a + 2 \cdot \frac{b-a}{2^{n+1}}\right) \right. \\ &\quad \left. + f\left(a + 3 \cdot \frac{b-a}{2^{n+1}}\right) + f\left(a + 4 \cdot \frac{b-a}{2^{n+1}}\right) + \cdots + \frac{f(b)}{2} \right\} \end{aligned} \quad (6.5)$$

ゆえに

$$T_{n+1} = \dots \quad (6.6)$$

よって分割数 N を増やしていくときに新たに出現する分点での関数の値だけを計算すればよい

6.1.2 台形則のアルゴリズム

(1)

$$\begin{aligned}\varepsilon &:= 10^{-p} \\ N &:= 1 \\ h &:= b - a \\ T &:= h(f(a) + f(b))/2\end{aligned}$$

(2) 以下繰り返し

$$\begin{aligned}N &:= 2N \\ h &:= h/2 \\ s &:= 0 \\ i &:= 1, 3, \dots, N-1 \text{ の順に, } s := s + f(a + ih) \text{ を繰り返す} \\ new_T &:= T/2 + h \cdot s \\ \text{もし, } |new_T - T| &< \varepsilon |new_T| \text{ ならば (3) に移る} \\ T &:= new_T\end{aligned}$$

(3) new_T を答えとする

6.2 シンプソン則

- 台形則は 1 次のラグランジュ補間多項式を利用し誤差 :
- 関数近似の精度を上げれば、積分精度が上がると期待される :
- 積分区間 $[a, b]$ を N 等分、分点間隔 : $h = (b - a)/N$
分点 : $x_j = a + jh \quad (j = 0, 1, 2, \dots, n)$
- シンプソン則 :
- 隣り合う 3 つの分点 : を含む区間 で関数 $f(x)$ を 2 次のラグランジュ補間 $P(x)$ で近似
- $P(x_j) = f(x_j), P(x_{j+1}) = f(x_{j+1}), P(x_{j+2}) = f(x_{j+2})$

$$P(x) = \frac{(x - x_j)(x - x_{j+1})}{(x_j - x_j)(x_j - x_{j+1})} + \frac{(x - x_j)(x - x_{j+2})}{(x_{j+1} - x_j)(x_{j+1} - x_{j+2})} + \frac{(x - x_{j+1})(x - x_{j+2})}{(x_{j+2} - x_j)(x_{j+2} - x_{j+1})} \quad (6.7)$$

- 積分を近似

$$x = x_j + th, \quad x_j = a + jh \text{ として} \quad (6.8)$$

$$\int_{x_j}^{x_{j+2}} f(x) dx \simeq \int_{x_j}^{x_{j+2}} P(x) dx \quad (6.9)$$

$$= \quad (6.10)$$

$$= \quad (6.11)$$

- 積分区間を
に分割（ゆえに偶数の分割）し、各区間
で、上記式を適用、合計をとる

$$S = \frac{h}{3} \{ f(x_0) + 4f(x_1) + f(x_2) \} + \\ + \quad (6.12)$$

$$= \frac{h}{3} \{ f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots \\ + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_{N-2}) \} \quad (6.13)$$

$f(x_0)$ と $f(x_N)$ の係数が 1, $f(x_1)$ から $f(x_{N-1})$ までの係数は

- シンプソン則の誤差：

6.2.1 シンプソン則と台形則の関係

- 終了条件は、台形則と同じ、分割数を $N = 2^n$ ($n = 1, 2, \dots$) で増やしていく、答え
が p 術変化しなくなったら、終了。
- 計算の効率化：台形則と同じく、ある n のときの計算を次の $n+1$ の計算で利用する。
- 台形則を利用した計算方法

$N = 2^n$ のときの台形則の積分近似 : T_n , シンプソン則の積分近似 : S_n

シンプソン則と台形則の関係 : $S_{n+1} =$ (6.14)

$$h = \frac{b-a}{2^{n+1}} \text{ とすると} \\ \frac{4}{3}T_{n+1} - \frac{1}{3}T_n \\ = \frac{4h}{3} \left\{ \frac{1}{2}f(a) + f(a+h) + f(a+2h) + \cdots + f(b-h) + \frac{1}{2}f(b) \right\} \\ - \frac{h}{3} \left\{ \frac{1}{2}f(a) + f(a+\quad) + f(a+\quad) + \cdots + f(b-2h) + \frac{1}{2}f(b) \right\} \quad (6.15)$$

※ T_n のとき、分割幅は $2h = \frac{b-a}{2^n}$ であることに注意

$$= \frac{h}{3} \{ f(a) + 4f(a+h) + 2f(a+2h) + \cdots + 2f(b-2h) + 4f(b-h) + f(b) \} \quad (6.16)$$

$$= S_{n+1} \quad (6.17)$$

6.2.2 シンプソン則のアルゴリズム

(1)

$$\varepsilon := 10^{-p}$$

$$N := 2$$

$$h := b - a$$

$$T :=$$

$$S :=$$

(2) 以下繰り返し

$$N := 2N$$

$$h := h/2$$

$$s := 0$$

$i := 1, 3, \dots, N - 1$ の順に, $s := s + f(a + ih)$ を繰り返す

$$new_T := T/2 + h \cdot s$$

もし, $|new_S - S| < \varepsilon |new_S|$ ならば (3) に移る

$$T := new_T$$

(3) new_S を答えとする

6.3 ロンバーグ積分法

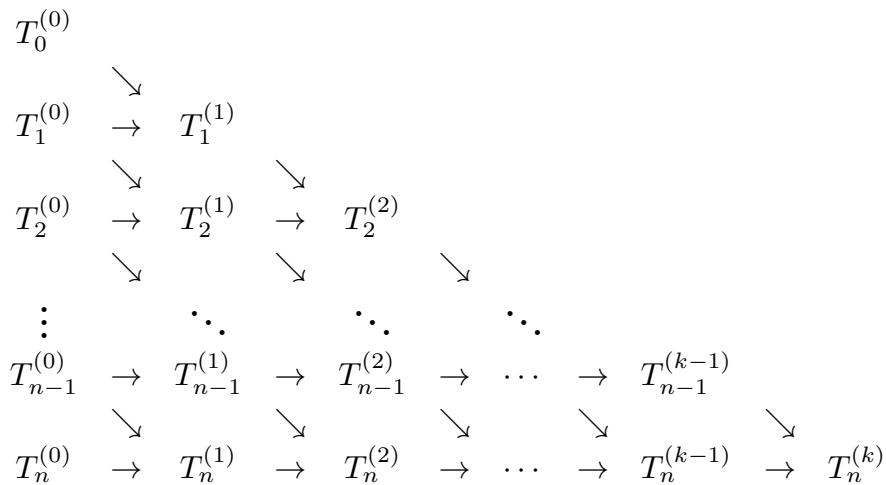
台形則とシンプソン則の関係は興味深い。なぜなら、誤差が $1/n^2$ に比例する台形則をもつて、分割数 N と $N+1$ での台形則の答えに、適当な係数をかけて差を取ると、誤差が $1/N^4$ に比例するシンプソン則が導かれる。このような方法で、さらに N を大きくしていくにつれて、もっと誤差をもっと速く小さくする方法が、ロンバーグ積分法である。

- 分割数 $N = 2^n \quad (n = 0, 1, 2, \dots)$
- 各 n に対して台形則によって計算された積分近似値 : $T_n^{(0)}$
- 漸化式 :

$$T_n^{(k)} = \frac{4^k T_n^{(k-1)} - T_{n-1}^{(k-1)}}{4^k - 1} \quad (6.18)$$

$$k=1 \text{ のとき, } T_n^{(1)} = \frac{4}{3}T_n^{(0)} - \frac{1}{3}T_{n-1}^{(0)} : \text{シンプソン則} \quad (6.19)$$

6.3.1 ロンバーグ積分法 $T_n^{(k)}$ を計算する手順



演習課題 レポート 7

関数 $\int_0^1 \frac{4}{1+x^2} dx = \pi$ の台形積分とシンプソン則による計算結果を比較するプログラム integrate1.py を実行し、下記を記してレポートを提出せよ。

- 1) 実行結果のグラフを載せること。
- 2) 実行結果のグラフが意味することを、述べ考察せよ。

(参考 URL) [Python による科学・技術計算] 数値積分、台形則・シンプソン則、数値計算、scipy

https://qiita.com/sci_Haru/items/09279cf81b9b073afa1d

```
# https://qiita.com/sci_Haru/items/09279cf81b9b073afa1d
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt

err_trape=[]
err_simps=[]
nn=[4,8,16,32,64,128,256,512,1024,2048] # 4 から 2048までのグリッド数をリスト nn に格納
for j in nn:
    x=np.linspace(0,1, j)
    y=4/(1+x**2)
    y_integrate_trape = integrate.cumtrapz(y, x) #台形則による数値積分計算
    y_integrate_simps = integrate.simps(y, x) #シンプソン則による数値積分計算
    err_trape.append(abs(np.pi-y_integrate_trape[-1])/np.pi) # 台形則による数値積分の相対誤差評価
    err_simps.append(abs(np.pi-y_integrate_simps)/np.pi) # シンプソン則による数値積分の相対誤差評価

# for plot
ax = plt.gca()
ax.set_yscale('log') # y 軸を log スケールで描く
ax.set_xscale('log') # x 軸を log スケールで描く
plt.plot(nn,err_trape,"-", color='blue', label='Trapezoid rule')
plt.plot(nn,err_simps,"-", color='red', label='Simpson rule')

plt.xlabel('Number of grids',fontsize=18)
plt.ylabel('Error (%)',fontsize=18)
plt.grid(which="both") # グリッド表示。"both"は xy 軸両方にグリッドを描く。

plt.legend(loc='upper right')

plt.show()
```