2005年度 修士論文

Toeplitz行列を係数行列にもつ大規模連立一次方程式の精度保証法

A numerical verification method for solutions of large-scale Toeplitz systems.

提出日： 2006年2月3日

指導： 大石 進一 教授

早稲田大学大学院 理工学研究科

情報・ネットワーク専攻 修士課程２年

学籍番号： 3604U137-2

福 地　健

目 次

表 目 次

図 目 次

第1章　序論

1.1　背景

数値計算が, 数学の応用上に重要な意味をもつことはいうまでもない. もちろん, 数学とは数値計算のことと思ったり, 逆に数値計算などは純粋数学者の関与するところではない, というような誤解ないしは偏見も, ないではなかったようである. もともと数値計算という分野は, 計算の精度と手間という相反する要求を調和させようとして生まれてきたものである. 理論的に美しい方法が案外役にたたず, 実際の問題を解く必要上, やむをえず考え出された苦しまぎれの方法が, かなりの部分をしめていることは事実である. そして, いずれにせよ, 実際に計算を実行するには, おびただしい手間と時間が必要であった. 対数表の発見, 計算機の発明など, そういった労苦の緩和に大きな役割を果たしたことはいうまでもないが, 科学の進歩につれて, 計算の労力は増加する一方であった.

近年になって, 電子計算機の異常なまでの急激な進歩により, こうした事情もかなり変わってきた. もちろん電子計算機といえども, 限度がある. 現存する世界一の計算機でも, まだ十分でない問題もある. しかし, かつては夢にすぎなかった大計算が, あいついで現実のものとなってきていることも事実である. もはや電子計算機を抜きにした数値計算論は, 過去のものとなったといってよいであろう.

しかしそのために, これまで問題にならなかった新しい種類の問題が発生してきた. 事実人間がみれば何でもなく解ける方程式を, 機械的に公式にあてはめて電子計算機にかけると, とんでもない答えがでてくる例もいろいろ報告されている. またこれまで実用には不適当だと考えられていた公式や計算法が, 新しく電子計算機のためにみなおされた例もある. なぜなら, 四則演算の結果はそのつど四捨五入に

よって丸められ, 極限を含む無限演算はすべて有限演算で近似して行われてしまうからだ. したがって計算結果は, 常にある程度の誤差をともなっている. そこで, 計算結果から正しい結論を得るためには, この計算結果に含まれている誤差を評価して真の値の範囲を確定することが必要となる.

これに対し, 真の値を含む区間を数とみなす区間解析の概念が導入され, 丸め誤差が存在する浮動小数点演算を用いても, 数学的に正しい結果を数値計算により導く原理が示された. この概念は活発に研究され, その結果いろいろな成果が得られたが, 一方では, 単に浮動小数点演算を区間演算に置き換えると, 多くの場合区間の幅が大幅に増大し, 意味ある結論が得られないことがわかった.

この困難を解消する方法として, 平均値形式などの関数の評価に微分係数の区間評価を用いる方法や, 近似解が得られた後にその周りで Newton 反復作用素に対して縮小写像原理が成り立つかどうかにより区間演算で数値的に検証する方法が示された. その結果, 真の解を含む区間を縮小させることも可能となり, 数値計算の誤差を厳密にかつシャープに評価することが原理的に可能であることがさまざまな例から明らかにされてきた.

この区間解析の例を始め, 近年能力が大幅に向上してきた計算機を利用することにより, 連続数学の問題の真の解の存在を証明したり, 真の解の誤差範囲が保証された精度のよい近似解を得る数値計算を精度保証つき数値計算という. 浮動小数点数演算における丸め誤差を含む演算結果の保証が理論的にも実用的にも高い精度で効率よく実現できることが明らかになり, 計算の信頼性という問題は, 数学としての数値解析の観点からようやく取り上げられることとなった. これは単に丸め誤差を厳密に評価するということだけにとどまらず, 科学技術計算の元となった数値解析アルゴリズムそのものに影響を与え, さまざまな数理科学上にあらわれる問題の解を, 数学的な厳密さで検証する方向にまで展開しつつある.

このように, 精度保証付き数値計算は計算の信頼性の立場から見た今後の科学技術の計算法のあるべき一つの方向として考えられようとしている. 近い将来, 数値計算に精度保証を付加するのが日常的になることも考えられる. いずれにせよ, 精

度保証付き数値計算の分野は, いま大きな転換期に面しており, 新しい飛躍が望まれているように思われる.

1.2 本論文の目的

現代の数値解析技術における精度保証付き数値計算の十分たる必要性は,1.1 節に述べた内容などからも明らかである. 例えば, ベクトルの内積計算や, 連立一次方程式の解を精度保証付きで求めることができるようになった.

本論文では, 有限次元線型方程式（連立一次方程式）の解の精度保証付き数値計算法に着目した. その中でも,Toeplitz 行列という特殊な形をした行列を対象に, その専用アルゴリズムを開発することによって, 従来よりもさらに高速に精度保証を行うことを目的としている.

実際にランダムな Toeplitz 行列を生成し, 精度保証のための定理を用いて精度保証を行い, その計算量の検証, 考察を行う. また,Toeplitz 行列の性質を 2 パターンに分け, それぞれに最適なアルゴリズムを用いることにより, 高速精度保証を行う.

1.3 本論文の構成

本論文の構成は以下の通りである.

第 2 章 「精度保証の準備」では, 精度保証付き数値計算を行うための準備として, 浮動小数点数システム, 区間演算, 精度保証の基本的概念について論じる.

第 3 章 「精度保証の方法」では, 連立一次方程式の近似解の数値計算法や, 精度保証のための定理について述べ, 連立一次方程式の解の精度保証法について論じる.

第 4 章 「Toeplitz 行列の逆行列およびその積の計算アルゴリズム」では,Toeplitz 行列を用いた連立一次方程式を高速化するためのアルゴリズムの概要について論

じる.

第 5 章 「高速精度保証の実装と結果」では, 実際に連立一次方程式を用いた解の精度保証付き数値計算を実装し, その結果を示す.

第 6 章 「結び」では, 本論文のまとめ, および結果の検証を行い, 新たな成果・問題点・今後の課題を記述する.

第2章　精度保証の準備

2.1　浮動小数点数

2.1.1　はじめに

本章では浮動小数点数とは何かを明らかにしていく. 浮動小数点数を利用して近似計算（数値計算）したり, その結果を検算（精度保証）するためである. 浮動小数点数システムについては,IEEE 標準 754(IEEE standard 754, 以降 IEEE754 と略記する) がパソコンやワークステーションなどをはじめとして多くのコンピューターで標準的に用いられている.

ここでは IEEE 推奨方式 754 に基づく 2 進数浮動小数点数システムを考えることにする. これは IEEE754 が理論的にも優れたシステムで, その骨格となる仮定を満たす他の浮動小数点数システムでも以下の議論は同様に成立するからである.

2.1.2　浮動小数点数

IEEE 754 では 4 つのタイプの数が用意されている. それは規格化 2 進浮動小数点数, 零, 非規格化 2 進浮動小数点数,NaN (Not a Number , 非数) である.

2 進規格化浮動小数点数

2 進規格化浮動小数点数とは

$$a = \pm \left(\frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_N}{2^N} \right) \times 2^{e-1023}, \quad (d_i = 0 \text{ または } 1) \tag{2.1}$$

と書ける数をいう.

$$m = \frac{d_1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_N}{2^N} \tag{2.2}$$

を符号付き仮数 (signed mantissa) といい,e を指数 (exponent) という. 指数 e も 2 進数の整数で表される. 通常, 浮動小数点システムには, 単精度 ($float$), 倍精度 ($double$), 拡張倍精度 ($long\ double$) があり, それぞれの bit 数の内訳は以下のとおりとなっている. なお, 上の指数部の値を e' とすると式 (2.1) の e

表 2.1: 浮動小数点システムにおける精度と bit 数の関係

精度	全長	符号	指数部	仮数部	指数部の $bios$
$float$	$32bit$	$1bit$	$8bit$	$23bit$	127
$double$	$64bit$	$1bit$	$11bit$	$52bit$	1023
$long\ double$	$80bit$	$1bit$	$15bit$	$64bit$	16383

の範囲は $1 \leq e \leq 2^{e'} - 2$ となっている. また,$e = 0, 2^{e'} - 1$ のときは次のように定められている.

$$e = 0, \quad m = 0 \quad \rightarrow a = 0 \tag{2.3}$$

$$e = 0, \quad m \neq 0 \quad \rightarrow \text{非正規化数} \tag{2.4}$$

$$e = 2047, \quad m = 0 \quad \rightarrow a = \pm\infty \tag{2.5}$$

$$e = 2047, \quad m \neq 0 \quad \rightarrow NaN \tag{2.6}$$

倍精度規格化 2 進浮動小数点システムにおいて表される数の絶対値の最大値

と最小値は

$$\begin{aligned} Max(\text{正規化数}) &= (1.111\cdots1)\times 2^{2046-1023} \\ &= 2^{1024}-2^{97} \\ &\cong 10^{308.25} \end{aligned} \tag{2.7}$$

$$\begin{aligned} Min(\text{正規化数}) &= (1.000\cdots0)\times 2^{1-1023} \\ &= 2^{-1022} \\ &\cong 10^{-307.65} \end{aligned} \tag{2.8}$$

である.

$|a| > Max(\text{正規化数})$ のときにオーバーフロー (overflow) が生じたという.

零

零は規格化されて, 式 (2.3) より

$$+\left(\frac{0}{2}+\frac{0}{2^2}+\frac{0}{2^3}+\cdots+\frac{0}{2^N}\right)2^0 \tag{2.9}$$

と表される.

非規格化 2 進浮動小数点数

式 (2.4) のようにIEEE 754 では浮動小数点数は指数部が 0 となったとき, 仮数部の最初の桁が 1 より小さい数を表すために, デフォルトで最初の桁を 1 とすることをやめ, ここが 0 となる数を置くことを許す規格となっている. これを非規格化数 (denormalized number) という. 非規格化数の範囲に数が入ることを, 漸近アンダーフロー (gradual underflow) という.

倍精度規格化 2 進浮動小数点システムにおいて表される数の絶対値の最大値

と最小値は

$$\begin{aligned} Max(\text{非正規化数}) &= (0.111\cdots1)\times 2^{-1022} \\ &= 2^{-1024}-2^{-1074} \\ &\cong 10^{-308.25} \end{aligned} \tag{2.10}$$

$$\begin{aligned} Min(\text{非正規化数}) &= (0.00\cdots01)\times 2^{-1022} \\ &= 2^{-1074} \\ &\cong 10^{-323.31} \end{aligned} \tag{2.11}$$

である.

$|a| < Min$(非正規化数) のときにアンダーフロー (underflow) が生じたという.

NaN

このほかに,IEEE 754 ではつぎのような特別な数が用意されている.

(a) 式 (2.3) で表される 0 はアンダーフローか $\pm\infty$ での割り算の結果として得られる.

(b) 式 (2.5) で表される $\pm\infty$ はオーバーフローの結果や零で割った結果として得られる.

(c) 式 (2.6) で表される NaN(Not a Number) は $\sqrt{-1}, \dfrac{\infty}{\infty}, \infty-\infty$ など不当な演算の結果として得られる.

マシンイプシロン (Machine Epsilon)

マシンイプシロン ϵ は,「(1 より大きな最小の浮動小数点数)-1」として定義され, その値は

$$\epsilon = (1.00\cdots01)\times 2^0 - 1 = 2^{-52} \cong 10^{-15.65} \tag{2.12}$$

となっている.

2.1.3 浮動小数点数と丸め

IEEE 754 では, つぎの 4 つの丸めのモードが指定できる.c を実数 ($c \in R$) とする.

上向きの丸め (round upward)

c 以上の浮動小数点数の中で最も小さい数に丸める. これを $\triangle : R \to F$ と表す. アルゴリズムでの表記は UP とする.

下向きの丸め (round downward)

c 以下の浮動小数点数の中で最も大きい数に丸める. これを $\nabla : R \to F$ と表す. アルゴリズムでの表記は DOWN とする.

最近点への丸め (round to nearest)

c に最も近い浮動小数点数に丸める. これを $\square : R \to F$ と表す. アルゴリズムでの表記は NEAR とする. もし, このような点が 2 点ある場合には, 仮数部の最後のビットが偶数である浮動小数点数に丸める. これを偶数丸め方式 (round to even) という.

切り捨て (round toward 0)

絶対値が c 以下の浮動小数点数の中で,c に最も近いものに丸める. アルゴリズムでの表記は ZERO とする.

2.1.4 浮動小数点数の性質

丸めの演算を写像として $\bigcirc : R \to F$ と書く. すなわち,$\bigcirc$ は $\triangle, \nabla, \square$ のいずれかと考える.IEEE 754 では, 丸めの演算は任意の $x \in R$ についてつぎの条件を満たす.

$$\bigcirc x = x$$

$$x \leq y \to \bigcirc x \leq \bigcirc y$$

また,$x \in F$ のとき, 符号を変えることにより,$-x$ や $|x|$ が得られるので, これらは正確に計算される. また,IEEE 754 では, 任意の $x \in R$ についてつぎの性質が成立する.

$$\begin{aligned} \square\,(-x) &= -\square\, x \\ \triangle(-x) &= -\triangle x \\ \nabla(-x) &= -\triangle x \end{aligned}$$

IEEE 754 では, 浮動小数点数演算 (F 上での四則演算) は丸めとの関係により, つぎのように定義されている. $\cdot \in \{+, -, \times, /\}$, $\bigcirc \in \{\triangle, \nabla, \square\}$ のとき

$$x \odot y = \bigcirc(x \cdot y), \quad (\text{任意の } x, y \in R \text{ について})$$

この式は, 左辺の浮動小数点数の四則演算の結果 $x \odot y$ は, 右辺の数学的に正しい (実数としての) 四則演算の結果 $x \cdot y$ を指定された丸めを行って得られた数 $\bigcirc(x \cdot y)$ に一致するように計算することを表している.

また, 平方根も $(\sqrt{x})_{fp} = \bigcirc(\sqrt{x})$, (任意の $x \in F$ について) と, 浮動小数点数演算によって計算された平方根 $(\sqrt{x})_{fp}$ は, 正確な実数演算で計算された平方根 $\sqrt{x}$ を指定された丸めの方向へ丸めた数となる. 注意すべきことは, 指数関数や三角関数などはこのような規格を満たしていない. つまり, 初等関数の値を精度保証付きで求めるためには工夫が必要なのである.

2.2 区間演算

2.2.1 はじめに

連続数学の問題で解が実数で与えられるものを考える. 浮動小数点数を用いた数値計算では, 真の解そのものを計算することは一般に不可能である. そこで, 真の解の下限を与える浮動小数点数と上限を与える浮動小数点数を計算する. もし, 上限と下限の数値が小数点以下 10 桁まで一致すれば, 真の解の小数点以下 10 桁まで

計算できたことになる. つまり, 精度保証付き数値計算の基本的な原理は浮動小数点数を両端とする区間の中に連続数学の問題の解を包み込むということである. この考え方は実数値で与えられる真の値の上限と下限を浮動小数点数により計算しようとするものであり, 区間解析 (interval analysis) と呼ばれ, 数学的理論と密接な関係がある.

区間解析では, 区間を数の拡張と考える. そして, その四則演算が定義される. これを区間演算という. 区間演算を浮動小数点数システムの上で展開するものを機械区間演算という. 機械区間演算の概念は, 精度保証付き数値計算の方法と密接に関連するものである.

2.2.2 四則演算の定義

まず, 実数上での演算を仮定する区間演算の概念を説明する. 区間解析において区間とは

$$[\underline{x}, \overline{x}] = \{x \in R | \underline{x} \leq x \leq \overline{x}\} \tag{2.13}$$

と表される閉区間であるとする. ただし, $\underline{x} \leq \overline{x} \in R$ で, それぞれ区間の下端, 上端とする. 以下, 記号の節約のため

$$[x] = [\underline{x}, \overline{x}] \tag{2.14}$$

と区間を表し, 区間 $[x]$ と単に書けば, それは $[x] = [\underline{x}, \overline{x}]$ を表すものとする.$\underline{x} = \overline{x}$ となる区間 $[x]$ を点区間 (point interval) という. 点区間は実数であるので, 点区間 $[x]$ の表す実数を x と書くことにする.

区間 $[x]$ について以下を定義する.

$$\begin{aligned} d([x]) &= \overline{x} - \underline{x} \\ r([x]) &= \frac{\overline{x} - \underline{x}}{2} \\ m([x]) &= \frac{\overline{x} + \underline{x}}{2} \end{aligned} \tag{2.15}$$

$d([x]), r([x]), m([x])$ をそれぞれ区間 $[x]$ の直径 (diameter), 半径 (radius), 中心 (center) という.

区間 $[x]$ の最小絶対値と最大絶対値をそれぞれつぎで定義する.

$$\begin{aligned} \langle [x] \rangle &= \min\{|x| \mid x \in [x]\} \\ |[x]| &= \max\{|x| \mid x \in [x]\} = \max\{|\underline{x}|, |\overline{x}|\} \end{aligned} \tag{2.16}$$

2 つの区間 $[x], [y]$ の距離を

$$q([x]) = \max\{|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|\} \tag{2.17}$$

で定義する.

2 つの区間 $[x], [y]$ が与えられたとき, その 2 つの区間の四則演算をつぎで定義する.

$$[x] \circ [y] = \{x \circ y \mid x \in [x], y \in [y]\} \tag{2.18}$$

ただし,$\circ \in \{+, -, \times, /\}$ である. これを区間演算 (interval arithmetic) という. この定義では, 区間演算は無限回の実数演算をしないと実行できないような印象を与えるが, 実はつぎが成立する.

$$\begin{aligned} [x] + [y] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\ [x] - [y] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\ [x] \times [y] &= [\min\{\underline{x}\underline{y}, \overline{x}\overline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\overline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}\}] \\ [x]/[y] &= [x] \times \left[\frac{1}{\overline{y}}, \frac{1}{\underline{y}}\right], \quad (0 \notin [y]) \end{aligned} \tag{2.19}$$

これから区間演算が有限回の実数演算で実行できることがわかる. 乗算 (表 2.2) と除算 (表 2.3) においては. 場合分けにより, より少ない手間で計算するつぎのような計算規則も簡単に導ける.

表 2.2: 区間 $[x], [y]$ の乗算 $[x][y]$

	$y \geq 0$	$y \ni 0$	$y \leq 0$
$x \geq 0$	$[\underline{xy}, \overline{xy}]$	$[\overline{x}\underline{y}, \overline{xy}]$	$[\overline{x}\underline{y}, \underline{x}\overline{y}]$
$x \ni 0$	$[\underline{x}\overline{y}, \overline{xy}]$	$[\min\{\underline{x}\overline{y}, \overline{x}\underline{y}\}, \max\{\underline{xy}, \overline{xy}\}]$	$[\overline{x}\underline{y}, \underline{xy}]$
$x \leq 0$	$[\underline{x}\overline{y}, \overline{x}\underline{y}]$	$[\underline{x}\overline{y}, \underline{xy}]$	$[\overline{xy}, \underline{xy}]$

表 2.3: 区間 $[x], [y]$ の除算 $[x]/[y]$

	$y \geq 0$	$y \leq 0$
$x \geq 0$	$[\underline{x}/\overline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\underline{y}]$
$x \ni 0$	$[\underline{x}/\underline{y}, \overline{x}/\underline{y}]$	$[\overline{x}/\overline{y}, \underline{x}/\overline{y}]$
$x \leq 0$	$[\underline{x}/\underline{y}, \overline{x}/\overline{y}]$	$[\overline{x}/\underline{y}, \underline{x}/\overline{y}]$

2.2.3 区間演算の性質

区間演算については, 包含関係における単調性

$$[x] \subseteq [x'],\ [y] \subseteq [y'] \Rightarrow [x] \circ [y] \subseteq [x'] \circ [y'], \quad (\circ \in \{+, -, \times, /\}) \tag{2.20}$$

が成立する. また, 加法と乗法に関し, 交換則と結合則が成立する.

$$\begin{aligned} &[x] \circ [y] = [y] \circ [x], \quad (\circ \in \{+, \times\}) \\ &[x] \circ ([y] \circ [z]) = ([x] \circ [y]) \circ [z], \quad (\circ \in \{+, \times\}) \end{aligned} \tag{2.21}$$

しかし, 加法と乗法の逆元は存在しない. すなわち, $-[x] = [-\overline{x}, -\underline{x}]$ であるが

$$\begin{aligned} &0 = [0] \subseteq [x] - [x] = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] \\ &1 = [1] \subseteq [x]/[x] \end{aligned} \tag{2.22}$$

となることがわかる. 上式で等号は $[x]$ が点区間のときのみ成立する.

また, 分配則も区間演算に対しては成立しない. そのかわりつぎの劣分配則が成立する.

$$[x] \times ([y] + [z]) \subseteq [x] \times [y] + [x] \times [z] \tag{2.23}$$

この式で等号は, 例えば区間 $[y]$ と $[z]$ が同じ符合をもつときに成立する.

2.2.4 区間拡張

関数 $f : D \subset R \to R$ を領域 D の任意の閉区間の上で連続な初等関数とする. 関数 f をつぎのようにして IR 上の関数に拡張することができる.

$$f([x]) = \{f(x) \mid x \in [x]\} \tag{2.24}$$

IR を実数上の有界閉区間の集合とする. 関数 $f : D \subset R \to R$ が初等関数の合成や四則演算に基づく演算規則で定義されているとき,f を IR 上の関数に拡張したい. しかし, $f([x])$ を厳密に計算することは非常な計算時間を要することがある. そこで, f の演算規則を単純に区間演算で置き換えた演算規則によって定義される関数を f の区間拡張 (interval extension) といい $f_{[\,]}$ で表す.

$$f([x]) \subseteq f_{[\,]}([x]) \tag{2.25}$$

が成立する. 区間拡張は一意的に定まるわけではなく, f を定義する数式を数学的に同等な他の表現形式に書き直したとき, それらの表現に基づく区間拡張が一般に異なる評価を与えることに注意する.

上式で等号が成立するとき, 厳密な包み込み (tight enclosure) ができたという. 厳密な包み込みは一般に難しいが, f の数学的な表現式に区間値パラメータが現れず, また,$[x]$ が一度しかその表現に現れなければ, その表現に基づく区間拡張が厳密な包み込みを与えることが知られている. 一般に式の表現に $[x]$ が少なく現れれば現れるほど, 区間包囲は厳密な包み込みに近くなることが多いことが経験されている. 区間 $[x]$ の幅 $d([x])$ が小さいとき

$$q(f([x]), f_{[\,]}([x])) \leq \alpha d([x]) \tag{2.26}$$

となることが知られている. ただし,α は正の $[x]$ によらない定数である.

なお, 一般に関数の値域 $f([a,b])$ を $f([a,b]) \subset [c,d]$ と区間 $[c,d]$ で評価するとき, 区間 $[c,d]$ のことを $f([a,b])$ の区間包囲という. 区間拡張は区間包囲の一種である.

2.2.5 区間の爆発

解を有限回の四則演算の組合せで求める方法 (直接解法という) において, 浮動小数点数を機械区間に置き換え, 四則演算を機械区間演算に置き換えて計算をすれば, 真の解を含む区間が得られる. こうして, 数値計算により数学的に正しい結論を導き出すために, 直接解法が知られている問題については, このような置き換えを行うことが盛んに行われた. ガウスの消去法において, 途中の計算における数を機械区間に, 四則演算を機械区間演算に置き換えたアルゴリズムを区間ガウス消去法という. 区間ガウス消去法はこのようなアルゴリズムの代表的なものである.

このようなアプローチはある程度の成功を収めたが, 反面, 機械区間演算を演算の単位と考えると, いろいろな問題が生じることもわかった. その一つは, このような計算では丸めの切替が頻繁におき, 計算速度の極端な低下が見られることや, プログラムが複雑になってしまうことである. また, 例えば, 区間ガウス消去法では, 実行の途中で現れる区間の幅の爆発が生じてしまうことが多い. したがって, 区間ガウス消去法は, 高々, 数十次元の線形方程式の解法としてのみ有効であると考えられている.

区間の爆発の例として

$$f(x) = x^2 + 2x \quad , \quad x \in [0.9, 1.1] \tag{2.27}$$

$$g(x) = x^2 - 2x \quad , \quad x \in [0.9, 1.1] \tag{2.28}$$

という 2 つの関数の値域の評価を考える. このとき関数 $f(x)$ の場合であれば

$$\begin{aligned} f(x) &= x^2 + 2x \quad , \quad x \in [0.9, 1.1] \\ &= [0.9, 1.1]^2 + 2 \times [0.9, 1.1] \\ &= [0.81, 1.21] + [1.8, 2.2] \\ &= [2.61, 3.41] \end{aligned} \tag{2.29}$$

となり, 実際の値域に一致する. しかし関数 $g(x)$ の場合だと

$$\begin{aligned} g(x) &= x^2 - 2x \quad , \quad x \in [0.9, 1.1] \\ &= [0.9, 1.1]^2 - 2 \times [0.9, 1.1] \\ &= [0.81, 1.21] - [1.8, 2.2] \\ &= [-1.39, -0.59] \end{aligned} \tag{2.30}$$

となり, 実際の値域 ([-1,-0.99]) とかなりかけ離れている. また, 区間の幅も実際の値域の 80 倍に膨れ上がっている. これが区間の爆発である.

区間の爆発が起こる理由として図 (2.1) と図 (2.2) を見てほしい ($x \in [0.9, 1.1]$ の部分を太線で表示してある).2 つのグラフはそれぞれ,$p(x) = x^2, x \in [0.9, 1.1]$ と $q(x) = 2x, x \in [0.9, 1.1]$ を表している. ここで区間の中間値である $x = 1$ のとき,$p'(1) = q'(1) = 2$ と計算できる. よって,2 つのグラフの $x \in [0.9, 1.1]$ の部分は傾きがともに 2 である直線とみなすことができる. このように $x \in [0.9, 1.1]$ において相関性のとても強いグラフになっているので, そのまま減算を区間演算で考えようとすると区間の爆発が起きてしまう可能性が高くなってしまうのである. これを回避するためには

$$g(x) = x^2 - 2x = (x-1)^2 - 1 \tag{2.31}$$

のように平方完成をして使用する文字数を減らすのが効果的である. この場合

$$\begin{aligned} g(x) &= (x-1)^2 - 1 \quad , \quad x \in [0.9, 1.1] \\ &= [-0.1, 0.1]^2 - 1 \\ &= [0, 0.01] - 1 \\ &= [-1, -0.99] \end{aligned} \tag{2.32}$$

となり実際の値域に合致する. この例のように与えられた式を変形させ, 変数の種類や数を減らしてから区間演算を実行したほうがより区間の幅を狭めることができることができ, より厳密な区間が得られることが多い.

実際に計算機上で精度保証をするときは, 丸めモードの指定の切り換えが必要となる. 例えば n 次元ベクトル x, y の内積を計算をする場合には以下のような計算をすれば良い.

$$\begin{aligned} \text{DOWN;} \quad & \underline{z} = \sum_{k=1}^{n} x_i \cdot y_i \\ \text{UP;} \quad & \bar{z} = \sum_{k=1}^{n} x_i \cdot y_i \end{aligned}$$

ここで, 命令 UP は上への丸めのモードへの切り換え, DOWN は下への丸めのモードへの切り換えを表している. なお, 最近点への丸めのモードの切り換えは NEAR で表す.

このような計算によって求められた $\underline{z}, \bar{z}$ によって, 真の解を含む区間 $[\underline{z}, \bar{z}]$ を得る. この区間をもって, 数値計算の計算結果の精度を保証しようというのが, 精度保証つき数値計算の基本的な概念である.

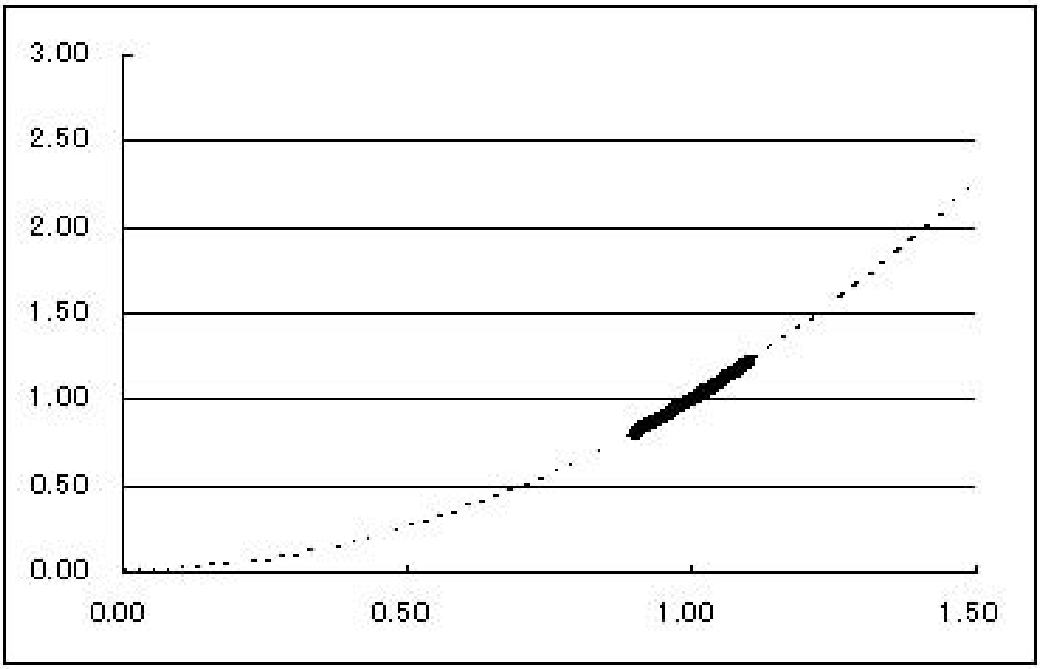


図 2.1: $p(x) = x^2, x \in [0.9, 1.1]$

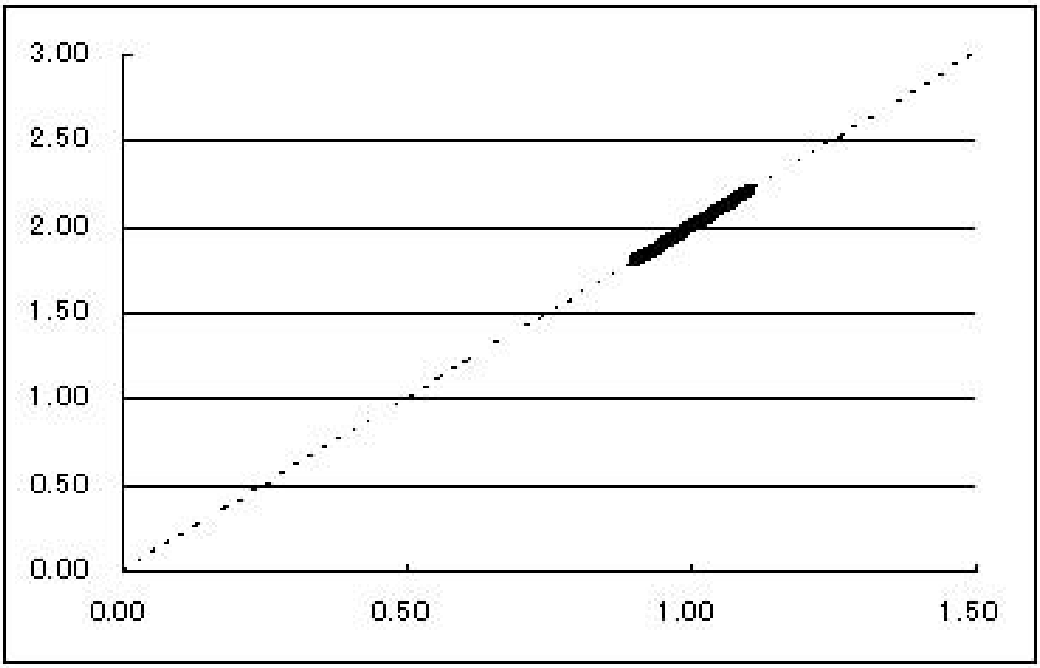


図 2.2: $q(x) = 2x, x \in [0.9, 1.1]$

第3章　精度保証の方法

3.1　はじめに

前章までで浮動小数点数演算の丸めの方向を制御することにより, ベクトルの内積計算などの精度を保証することが容易に行えることを示した. この章では, これを基礎に, 以下のような有限次元線型方程式（連立一次方程式）

$$Ax = b \tag{3.1}$$

の解の精度保証付き数値計算法を考えることにする. ただし A を $n \times n$ 行列,x や b を n ベクトルとする. 行列 A と右辺ベクトル b が与えられたとして, ベクトル x を求めるのが問題である.

3.2　近似解の数値計算法

連立一次方程式の近似解を得る数値解法としてよいと考えられているものにはいくつかある. その中で最も重要でよく用いられているのは LU 分解法である.LU 分解法では

$$PA = LU \tag{3.2}$$

と入力の係数行列 A を下三角行列 L と上三角行列 U の積に分解する. ただし

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \tag{3.3}$$

である. 行列 L は行列の下半分のみが非零な下三角行列 (lower triangular matrix) の一種である. また

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \ddots & u_{3n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & u_{nn} \end{bmatrix} \tag{3.4}$$

である. 行列 U は行列の上半分のみが非零な上三角行列 (upper triangular matrix) の一種である. さらに,P は置換行列で, 各行各列に 1 が一つずつあり, 他は 0 の行列である. これは, 行と行を入れ替える操作を表している. そして

$$Ly = Pb \tag{3.5}$$

を解いて y を求め, つぎに

$$Ux = y \tag{3.6}$$

を解いて x を求める方法である. 容易にわかるように, この二つの方程式はともに三角行列を係数行列にしているので, 簡単に代入操作で解ける. このように, 行列 A を下三角行列 L と上三角行列 U の積に分解することで, 連立一次方程式の近似解を求める方法を LU 分解法と呼ぶ.

LU 分解を求めるアルゴリズムにはいろいろあるが, ガウスの消去法はよく知られている. 行列 A に $n-1$ 回の行操作を行って,A の LU 分解を求めるというもので

ある. 消去がうまく進むように, 行の変換を行う操作を部分ピボッティングという. ピボッティングの結果は置換行列 P に書き込まれる.

行列 A の逆行列を計算する一つの方法は,LU 分解の利用である. 行列 A の逆行列を $R = A^{-1}$ とする. ここで

$$R = (r_1, r_2, \cdots, r_n) \tag{3.7}$$

と表す. ただし,r_i は逆行列 R の第 i 列を表す n ベクトルとする. また,e_i を第 i 成分が 1 でその他の成分が 0 の n ベクトルとする.

$$Ar_i = e_i \tag{3.8}$$

であるから,A の LU 分解を求め, 上式を $i = 1, 2, \cdots, n$ と n 回解けば逆行列が求められることになる. その乗除算の演算回数は $n^3 + O(n^2)$ 回である. また, ガウス・ジョルダン法などの算法も知られている.

$n \times n$ 行列 A が与えられた時, これを LU 分解して

$$Ax = b \tag{3.9}$$

の解 x_0 を求めたとしよう. 丸め誤差がなければ,x_0 は真の解であるが, 実際には, 丸め誤差のために x_0 は真の解とはなっていない. 真の解を x^* とし,x_0 との誤差を $\epsilon = x_0 - x^*$ とする. このとき

$$\|x_0 - x^*\| = \|A^{-1}(Ax_0 - Ax^*)\| = \|A^{-1}(Ax_0 - b)\| \tag{3.10}$$

となる. 残差 $r = Ax_0 - b$ を用いて書けば,

$$\|\epsilon\| \leq \|A^{-1}r\| \tag{3.11}$$

となる. 浮動小数点計算では, 誤差が真の解に比べてどのくらいの比だけあるかということが, 問題になること多いので, 相対誤差 $\|\epsilon\|/\|x^*\|$ を考える.

$$\|b\| = \|Ax^*\| \leq \|A\|\|x^*\| \tag{3.12}$$

より,

$$\frac{\|\epsilon\|}{\|x^*\|} = \frac{\|A^{-1}r\|}{\|x^*\|} \leq \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|} \tag{3.13}$$

を得る.

$$cond(A) = \|A\|\|A^{-1}\| \tag{3.14}$$

を条件数 (condition number) と呼ぶ.

$$\|x\| \leq \|AA^{-1}x\| \leq \|A\|\|A^{-1}\|\|x\| \tag{3.15}$$

より $cond(x) \geq 1$ がわかる. このことと式 (3.13) から, 条件数は残差が何倍拡大されて相対誤差に反映するかの倍率となっていることがわかる. 倍精度浮動小数点数の仮数部の精度は,10 進数にして 16 桁程度であるから, 条件数が 10^{16} を超えると, 得られた近似解の精度はほとんど 1 桁もないことになる. すなわち, 条件数が 10^{15} くらいまでの問題が特別の工夫をしないで解ける線形問題の最大の条件数ということになる.

3.3 逆行列を用いた精度保証

方程式

$$Ax = b \tag{3.16}$$

の近似解 $\tilde{x}$ が与えられた時に, その近くに真の解が存在するか否かの判定や真の解との誤差を求める方法を示していく.

3.3.1 精度保証のための定理

連立一次方程式の近似解の精度保証を行うための原理となる定理を下に示す.

定理 3.1

$\tilde{x}$ と A の逆行列 R が求められた時, 不等式

$$\|RA - I\| < 1 \tag{3.17}$$

を満たす時は, 逆行列 A^{-1} が存在し

$$\|A^{-1}\| \leq \frac{\|R\|}{1 - \|RA - I\|} \tag{3.18}$$

および x^* を真の解として

$$\|x^* - \tilde{x}\| \leq \frac{\|R(b - A\tilde{x})\|}{1 - \|RA - I\|} \leq \frac{\|R\|\|(b - A\tilde{x})\|}{1 - \|G\|} \tag{3.19}$$

が成り立つ.

連立一次方程式 $Ax = b$ に対して,A の近似逆行列 R を求め, 近似解 $x = Rb$ を計算する. このとき, この式の真の解の存在の十分条件を検証し, もし存在する場合には, 真の解 x^* と x との間の誤差, つまり $\| x - x^* \|_\infty$ を, 上記の定理に基づいて計算する.

それでは実際に前述の定理に基づいて, 解 x の係数行列が Toeplitz 行列のとき, 通常の場合に比べより高速に精度保証する手順について述べる. 近似解 x が求められたとき, その近くに真の解 x^* が存在するか否か, また真の解と近似解との間の誤差を評価（精度保証）する.

ここでもう一度定理3.1 を評価してみる. この式の後項の計算には $R, b-A\tilde{x}, RA-I$ のそれぞれを計算しなければならないが, ここで $b - A\tilde{x}$ に着目すると, ベクトル×行列なので計算量が $O(n^2)$ で済むことは明らかである. 要するに,

- 係数行列の逆行列である R の計算
- その積である RA の計算

が重要になってくる. 次章では, この二つの計算方法について述べる.

3.3.2 精度保証のための定理の証明

定理 3.1 は, 関数解析におけるバナッハの縮小写像定理（特に, その系であるバナッハの摂動定理）からただちに導かれるものなのだが, ここでは, もう少し直接的な証明を示す.

定理 3.1 の証明

$r = A\tilde{x} - b$ を残差とする.$\|RA - I\| < 1$ のとき,$I + (RA - I)$ の逆行列が存在することが示せる.$G = RA - I$ とする.$\|G\| < 1$ のとき,$I + G$ の逆行列が存在することはつぎのようにしてわかる. 今,$I + G$ が特異であるとする. すると, あるベクトル $x \neq 0$ が存在して $(I + G)x = 0$ を満たす.$\|x\| = 1$ であるように規格化されているとしても一般性を失わない. このとき,

$$Gx = -x, \|x\| = 1 \tag{3.20}$$

が成立する. これは,$\|Gx\| = \|x\| = 1$ から,$\|Gx\| \geq 1$ を意味している. これは, 仮定と矛盾する. よって $I + G$ は正則である. さて,$I + G$ が正則であることから

$$I = (I + G)^{-1}(I + G) \tag{3.21}$$

となる. これを書き換えると

$$(I + G)^{-1} = I - (I + G)^{-1}G \tag{3.22}$$

となる. よって

$$\|(I + G)^{-1}\| \leq \|I\| + \|(I + G)^{-1}\|\|G\| \tag{3.23}$$

を得る.$\|G\| < 1$ より

$$\|(I + G)^{-1}\| \leq \frac{1}{\|1 - G\|} \tag{3.24}$$

を得る. 次のような評価を行う.

$$
\begin{aligned}
&\|A^{-1}\| \\
&= \|A^{-1}R^{-1}R\| \\
&= \|(I + (RA - I))^{-1}R\| \\
&\leq \frac{\|R\|}{1 - \|RA - I\|}
\end{aligned} \tag{3.25}
$$

同様に

$$
\begin{aligned}
&\|x^* - \tilde{x}\| \\
&= \|A^{-1}R^{-1}Rr\| \\
&= \|(I + (RA - I))^{-1}Rr\| \\
&\leq \frac{\|Rr\|}{1 - \|RA - I\|}
\end{aligned} \tag{3.26}
$$

これは所望の結果である. ただし,$r = A\tilde{x} - b$ とする.

Oishi-Rump の方法では IEEE 754 規格に定められている丸めの制御を適宜変更してこの誤差評価式の上限が計算される. この手法を用いると精度保証は近似解を求めるのに要する時間の数倍の手間で行うことができる. また係数行列が特殊な構造を持つ場合はより高速に精度保証ができることも示されてきた. 例えば Rump と Ogita は係数行列に対称正定値行列を持つ連立一次方程式の精度保証はコレスキー分解の後,$O(n^2)$ の計算量で行えることを示した. また Ogita,Oishi,Ushiro は M 行列などの単調行列を係数行列に持つ場合は $O(n^2)$ の計算量で精度保証が行えることを示した. このように係数行列が特殊な構造を持つ場合の連立一次方程式の精度保証は特殊な構造を持たない一般の行列を係数行列とする場合よりも高速に行えることが望まれる.

Toeplitz 行列を係数行列に持つ連立一次方程式は信号処理や画像復元などの分野に現れ,Levinson-Durbin によるアルゴリズムにより $O(n^2)$ の計算量で近似解を求めることができる. また Toeplitz 行列を格納するためのメモリ量は $O(n)$ である

が，精度保証に用いる近似逆行列は$O(n^2)$のメモリ量を必要とする. このため大規模な連立一次方程式の近似解を求めることが可能であっても近似逆行列を作成するメモリ領域を確保できないために精度保証が行えない可能性がある. そこで本論文では係数行列にToeplitz行列を持つ連立一次方程式に対して, 精度保証の際に使用するメモリ量を$O(n)$に抑える, 大規模行列に対応できる精度保証法を提案し, 数値実験により提案手法の有効性を示す.

第4章　Toeplitz行列の逆行列およびその積の計算アルゴリズム

本章ではToeplitz行列を係数行列に持つ連立一次方程式の精度保証法を係数行列が対称の場合と非対称の場合に分けて提案する. またToeplitz行列は$O(n)$のメモリ量があれば格納できるが, 精度保証で必要な逆行列を格納するために必要なメモリ量は$O(n^2)$である. このため近似解を得ることができる計算機環境において, 逆行列を格納するのに必要なメモリが不足するために精度保証ができないことがある. そこで精度保証に必要なメモリ量を$O(n)$に抑え, 大規模な連立一次方程式に対しても精度保証が行える手法を提案する.

4.1　Toeplitz行列の特徴とその応用例

本論文で扱うToeplitz行列について述べておく. 簡単に言えばToeplitz行列とは, 対角要素が定数となる行列のことである. 具体的には次のような形をしている.

$$A = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n-1} \\ a_{-1} & a_0 & a_1 & \ddots & \vdots \\ a_{-2} & a_{-1} & a_0 & \ddots & a_2 \\ \vdots & \ddots & \ddots & \ddots & a_1 \\ a_{-(n-1)} & \cdots & a_{-2} & a_{-1} & a_0 \end{bmatrix} \tag{4.1}$$

ここで, 行列の要素$a_k (k = -n+1, \ldots, -1, 0, 1, \ldots, n-1) \in \mathbb{F}$は全部で$2n-1$個あり, この要素が対称的に並ぶことによって$n \times n$行列を作り上げている. なお,$\mathbb{F}$を浮動小数点数の集合とする.$n \times n$行列といっても, 要素の数が$2n-1$個だけで済

むのでその分計算機上での記憶量は少ない. この Toeplitz 行列, 広い意味で Toeplitz 構造と言い方を変えるが, あらゆる統計的な信号処理ツールの開発において現れることが多い.

スペクトル推定法を含む信号処理ツールを例に挙げよう. スペクトル推定の目的は, 有限長のデータをベースに, 信号内に含まれているエネルギーの分布 (周波数上) を記述することである. パワースペクトルの推定は. 広い帯域に分布する雑音の中に含まれている信号の検出を含む, 種々のアプリケーションで有効となっている. 信号の長さが短い場合, パラメトリック推定法は, ノンパラメトリック推定法よりも高い解像度を提供する. 二つの方法は, スペクトル推定へのアプローチが異なるもので, データから直接 PSD を推定する代わりに, データを白色雑音がある線形システムに入力した結果の出力としてモデル化するもので, 結果としてその線形モデルのパラメータを推定するものである.

一般的に使用される線形システムモデルは, 全極モデルで,z 平面の原点にすべての零点をもつフィルタである. このようなフィルタへの白色雑音の出力は, 自己回帰 (AR) プロセスであることから, これらの方法はスペクトル推定の AR モデルと見なされる.

AR モデルは, データの PSD がある周波数で大きくなる, すなわち,"ピーク"を示すようにデータのスペクトルを適切に記述する傾向がある. 多くの実際的なアプリケーションの中のデータは,"ピークスペクトル"を示す傾向があり,AR モデルは非常に有効になる. 加えて,AR モデルは比較的簡単に解くことのできる線形方程式を導く.AR 法はいくつか存在するが, すべての AR 法は次式で得られる PSD を与える.

$$\hat{P}_{AR}(f) = \frac{1}{f_s} \frac{\epsilon_p}{\mid 1 + \sum_{k=1}^{p} \hat{a}_p(k) e^{-2\pi j k f / f_L} \mid^2} \tag{4.2}$$

AR 法のそれぞれは,AR パラメータ $a_p(k)$ をわずかに異なる値で推定する. その中でも, スペクトル推定の Yule-Walker AR 法は, 信号の自己相関関数のバイアス付き推定を作成し, 前置予測誤差の二乗が最小になるように解き,AR パラメータを計算する. このことを,Yule-Walker 方程式で表わすことができる.

$$\begin{bmatrix} r(1) & r(2)^* & L & r(p)^* \\ r(2) & r(1) & L & r(p-1)^* \\ M & 0 & 0 & M \\ r(p) & L & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ M \\ a(p+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ M \\ -r(p+1) \end{bmatrix} \tag{4.3}$$

自己相関関数のバイアス付き推定の使用は, 上の自己相関行列が正定である必要がある. そのため行列が可逆で, 解が存在することは保証される. さらに, 計算されるAR パラメータは, 安定な全極モデルに常になる.Yule-Walker 方程式は,Levinson アルゴリズムを使って, 効率的に解くことができる. これは, 自己相関行列の Toeplitz 構造を利用している.

4.2 Symmetric Matrices

4.2.1 Persymmetric Matrices とは

以下では, 正定値である Toeplitz 行列に対して Durbin のアルゴリズムを適用し話を進めるので, 先に Persymmetric Matrices と正定値について説明する.Persymmetric Matrices とは, すべての i, j に対して $b_{ij} = b_{n-j+1,n-i+1}$ をみたす行列を指し, 言い換えると,$B_n = P_n B_n^t P_n \in \mathbb{F}^{n\times n}$ とも表せる. ここで $P_n \in \mathbb{F}^{n\times n}$ は行または列を入れ替える操作に相当する対称行列である. また,Toeplitz 行列はすべて

Persymmetric Matrices である.

$$
\begin{aligned}
B_n &= \begin{bmatrix}
b_{11} & b_{12} & \dots & b_{1,n-1} & b_{1,n} \\
b_{21} & b_{22} & \dots & \ddots & b_{2,n} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
b_{n-1,1} & \ddots & \ddots & b_{n-1,n-1} & b_{n-1,n} \\
b_{n,1} & b_{n,2} & \dots & b_{n,n-1} & b_{n,n}
\end{bmatrix} \\
&= \begin{bmatrix}
b_{11} & b_{12} & \dots & b_{1,n-1} & b_{1,n} \\
b_{21} & b_{22} & \dots & \ddots & b_{1,n-1} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
b_{n-1,1} & \ddots & \ddots & b_{22} & b_{12} \\
b_{n,1} & b_{n-1,1} & \dots & b_{21} & b_{11}
\end{bmatrix}
\end{aligned} \tag{4.4}
$$

$$
P_n = \begin{bmatrix}
0 & 0 & \dots & 0 & 1 \\
0 & 0 & \dots & 1 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 1 & \dots & 0 & 0 \\
1 & 0 & \dots & 0 & 0
\end{bmatrix} \tag{4.5}
$$

また正定値であるとは, 以下のような二次形式に対し, 以下の Q の値が常に正になるときを言う.

$$
Q = x^t S x \in \mathbb{F}^{n\times n}, S \in \mathbb{F}^{n\times n}, x \neq 0, x \in \mathbb{F}^n \tag{4.6}
$$

4.2.2 Symmetric Matrices とは

次に Symmetric Matrices（対称行列）について説明する.Symmetric Matrices とは正方行列 $C_n \in \mathbb{F}^{n\times n}$ のうち C_n の転置行列 C_n^t が C_n 自身と一致するものであり,

すべての i, j に対して $c_{ij} = c_{ji}$ を満たす行列を指す.

$$
\begin{aligned}
C_n &= \begin{bmatrix}
c_{11} & c_{12} & c_{13} & \cdots & c_{1,n} \\
c_{21} & c_{22} & \cdots & \ddots & c_{2,n} \\
c_{31} & \vdots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & c_{n-1,n-1} & c_{n-1,n} \\
c_{n,1} & c_{n,2} & \cdots & c_{n,n-1} & c_{n,n}
\end{bmatrix} \\
&= \begin{bmatrix}
c_{11} & c_{12} & c_{13} & \cdots & c_{1,n} \\
c_{12} & c_{22} & \cdots & \ddots & c_{2,n} \\
c_{13} & \vdots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & c_{n-1,n-1} & c_{n-1,n} \\
c_{1,n} & c_{2,n} & \cdots & c_{n-1,n} & c_{n,n}
\end{bmatrix}
\end{aligned}
\tag{4.7}
$$

4.2.3 Durbin アルゴリズム

まず,Yule-Walker 方程式を解く際に利用される Durbin アルゴリズムを評価する.Yule-Walker 方程式とは, 次のような形をした方程式のことである.

$$
T_{n-1}y = -r = -(r_1, \ldots, r_{n-1})^t \tag{4.8}
$$

ここで,$T_n \in \mathbb{F}^{n\times n}$ は n 次対称 Toeplitz 行列,$y \in \mathbb{F}^{n-1}, r \in \mathbb{F}^{n-1}$ である. この方程式の解を帰納的に導き出すアルゴリズムが Durbin アルゴリズムである. 行列 T_n を, 次のような成分に分けて方程式を組みなおす. なお,$z \in \mathbb{F}^{n-1}, \alpha \in \mathbb{F}$ とする.

$$
\begin{bmatrix} T_{n-1} & P_{n-1}r \\ r^tP_{n-1} & 1 \end{bmatrix} \begin{bmatrix} z \\ \alpha \end{bmatrix} = - \begin{bmatrix} r \\ r_n \end{bmatrix} \qquad \left(y^{(n)} = \begin{bmatrix} z \\ \alpha \end{bmatrix} \right) \tag{4.9}
$$

これより, 以下の 2 式が求められる.

$$\begin{aligned} z &= T_{n-1}^{-1}(-r - \alpha P_{n-1} r) \\ &= -T_{n-1}^{-1} r - \alpha T_{n-1}^{-1} P_{n-1} r \\ &= -T_{n-1}^{-1} r - \alpha P_{n-1} T_{n-1}^{-1} r \\ &= y + \alpha P_{n-1} y \ (\because (4.8)) \end{aligned} \tag{4.10}$$

$$\begin{aligned} \alpha &= -r_n - r^t P_{n-1} z \\ &= -r_n - r^t P_{n-1} y - \alpha r^t y \ (\because (4.10)) \\ &= -(r_n + r^t P_{n-1} y)/(1 + r^t y) \end{aligned} \tag{4.11}$$

ただ, このアルゴリズムでは, ベクトル y の要素を完成させるのに $3n^2 flops$ の計算量が必要になる. しかし, さらに式変形を行い,$1 + r^t y$ を以下のように表すことによって計算量を少なくすることができる.

$$\begin{aligned} \beta_k &= 1 + [r^{(k)}]^t y^{(k)} \\ &= 1 + \begin{bmatrix} [r^{(k-1)}]^t & r_k \end{bmatrix} \begin{bmatrix} y^{(k-1)} + \alpha_{k-1} P_{k-1} y^{(k-1)} \\ \alpha_{k-1} \end{bmatrix} \ (\because (4.9)) \\ &= 1 + [r^{(k-1)}]^t y^{(k-1)} + \alpha_{k-1}([r^{(k-1)}]^t P_{k-1} y^{(k-1)} + r_k) \\ &= \beta_{k-1} + \alpha_{k-1}(-\beta_{k-1}\alpha_{k-1}) \\ &= (1 - \alpha_{k-1}^2)\beta_{k-1} \end{aligned} \tag{4.12}$$

この条件をもとに k を 1 から $n-1$ まで帰納的に計算することによって解が求まる.

Durbin アルゴリズム　　$r \in \mathbb{F}^n$ に対して,Yule-Walker 方程式 $T_n y = -r$ の近似解 $y \in \mathbb{F}^n$ を求める $Durbin$ によるアルゴリズム.

function y =Solve_Toeplitz(r)

$y(1) = -r(1); \beta = 1; \alpha = -r(1)$

for $\quad k = 1 : n-1$

$\beta = (1-\alpha^2)\beta$

$\alpha = -(r(k+1) + r(k:-1:1)^t y(1:k))/\beta$

$z(1:k) = y(1:k) + \alpha y(k:-1:1)$

$$y(1:k+1) = \begin{bmatrix} z(1:k) \\ \alpha \end{bmatrix}$$

end

このアルゴリズムは $2n^2 flops$ の計算量で実行できるが, 正定値ではない係数行列を持つ連立一次方程式に対しては 0 による除算が発生したり, 精度の良い近似解が得られない可能性があることに注意する.

4.2.4 逆行列の導出

以下で, 上記のアルゴリズムより得られた結果を用いて逆行列を求める工程を示す. まず, 行列 T_n に対する近似逆行列を R_n とすると,$T_n R_n = I_n$ ゆえ, それを次のように表すことができる.

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} B_{n-1} & v \\ v^t & \gamma \end{bmatrix} = I_n \tag{4.13}$$

ここで

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} v \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.14}$$

より, 以下の式を導くことができる.

$$\begin{aligned} v &= -\gamma T_{n-1}^{-1} P_{n-1} r \\ &= -\gamma P_{n-1} T_{n-1}^{-1} r \\ &= \gamma P_{n-1} y \ (\because (4.8)) \end{aligned} \tag{4.15}$$

$$\begin{aligned} \gamma &= 1 - r^t P_{n-1} v \\ &= 1 - \gamma r^t y \ (\because (4.8)) \\ &= 1/(1 + r^t y) \end{aligned} \tag{4.16}$$

これにより,T_n^{-1} の一番最後の行と列が求められたということになる. また,

$$\begin{bmatrix} T_{n-1} & P_{n-1} r \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} B_{n-1} \\ v^t \end{bmatrix} = \begin{bmatrix} I_{n-1} \\ 0 \end{bmatrix} \tag{4.17}$$

より, 以下の式を得る.

$$\begin{aligned} B_{n-1} &= T_{n-1}^{-1}(I_{n-1} - P_{n-1} r v^t) \\ &= T_{n-1}^{-1} - T_{n-1}^{-1} P_{n-1} r v^t \\ &= T_{n-1}^{-1} - P_{n-1} T_{n-1}^{-1} r v^t \\ &= T_{n-1}^{-1} + P_{n-1} y v^t \ (\because (4.8)) \end{aligned} \tag{4.18}$$

ここで,T_{n-1} は Toeplitz 行列であり, その逆行列は対称行列となるから, 行列 B_{n-1} の要素 b_{ij} は次のように表せる.

$$\begin{aligned} B_{ij} &= (T_{n-1}^{-1})_{ij} + P y_i v_j \\ &= (T_{n-1}^{-1})_{n-j,n-i} + P y_i v_j \\ &= b_{n-j,n-i} - y_{n-j} v_{n-i} + y_i v_j \end{aligned} \tag{4.19}$$

上式より行列 B_n は対称行列ではないが,T_n^{-1} の対称性から, 要素 b を行列の外側から導き出すことができる.

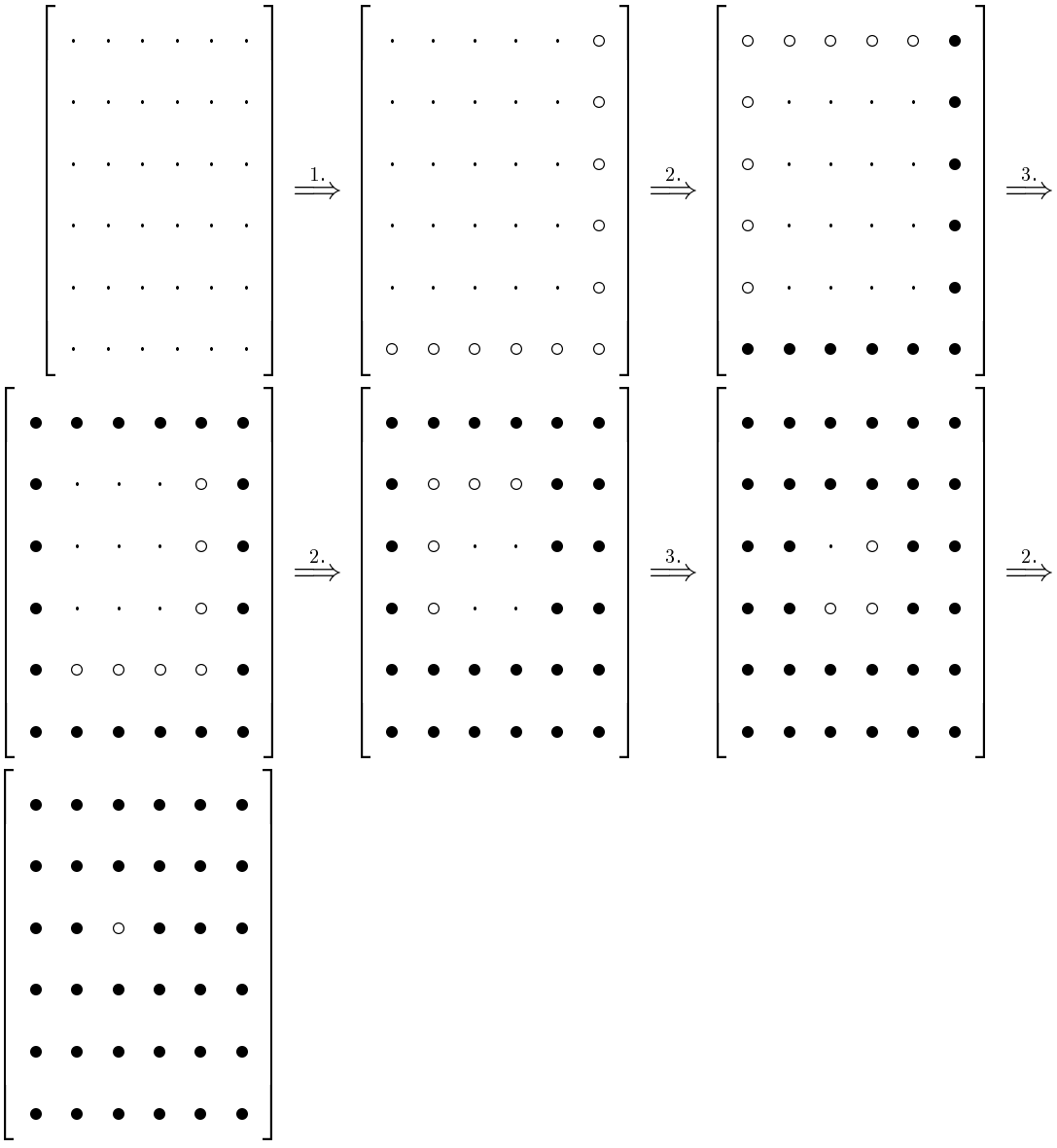
今までの議論より,T_n の近似逆行列 R_n は次の手順を繰り返すことによって求めることができるので,R_n は渦巻き状に決定していくことになる.

1. (4.15)(4.16) より,R_n の一番最後の行と列の要素を求める.

2. 行列の対称性から一番最初の行と列の要素を求める.

3. (4.19) より,R_{n-1} 内の一番最後の行と列を求める.

以下に, この手順を使った逆行列作成のシュミレート結果を挙げておく.
なお,[•] は既知の要素,[○] は新しく判明した要素,[·] は未知の要素を表すものとする.



また,R_n は完全に対称な行列なので, 全ての要素を求めなくともよい. これより, 逆行列 R_n を求めるアルゴリズムは次のように表される.

Trench アルゴリズム　$r \in \mathbb{F}^{n-1}$ によって定義された対称 Toeplitz 行列 $T \in \mathbb{F}^{n\times n}$ に対する近似逆行列 $R_n \in \mathbb{F}^{n\times n}$ を求める Trench によるアルゴリズム.

function R_n =Toeplitz_Inverse(r)

$y = Solve\_Toeplitz(r(1:n-1))$

$\gamma = 1/(1 + r(1:n-1)^t y(1:n-1))$

$v(1:n-1) = \gamma y(n-1:-1:1)$

$B_n(1,1) = \gamma$

$B_n(1,2:n) = v(n-1:-1:1)^t$

for　$i = 2 : floor((n-1)/2) + 1$

　　for　$j = i : n - i + 1$

$$B_n(i,j) = B_n(i-1,j-1) + (v(n+1-j)v(n+1-i) - v(i-1)v(j-1))/\gamma$$

　　end

end

このアルゴリズムは $13n^2/4 flops$ の計算量で実行でき, 近似逆行列の対称性と persymmetric 性を利用し行列全体の $1/4$ の成分を計算する. この手法では Durbin によるアルゴリズムを用いるため係数行列が正定値ではない場合は精度の良い近似逆行列が得られない可能性があり, 精度保証が失敗することがある.

4.3 Unsymmetric Matrices

4.3.1 Levinson-Durbin アルゴリズム

さて, 今までの議論は前提として対称 Toeplitz 行列のケースに限って行われてきたものだが, 本節では非対称 Toeplitz 行列の場合も同様なことが言えるかどうかについて述べる.

節 (4.2.4) を延長した形として, もう一つのアルゴリズムがある. このアルゴリズムによって, 右辺に任意のベクトルを持つ方程式を解くことができる.$x, b \in \mathbb{F}^{n-1}$ として次の方程式を解いてみる.

$$T_{n-1}x = b = (b_1, \ldots, b_{n-1})^t \tag{4.20}$$

この方程式の解を帰納的に導き出すアルゴリズムが Levinson アルゴリズムである. 節 (4.2.3) で論じた Durbin アルゴリズムと同様に考えると,$T_n x^{(n)} = b^{(n)}$ は次のように表すことができる. なお,$\mu \in \mathbb{F}$ とする.

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} v \\ \mu \end{bmatrix} = \begin{bmatrix} b \\ b_n \end{bmatrix} \tag{4.21}$$

ここで,$T_{n-1}v + \mu P_{n-1}r = b$ より,

$$\begin{aligned} v &= T_{n-1}^{-1}(b - \mu P_{n-1}r) \\ &= x - \mu T_{n-1}^{-1} P_{n-1} r \\ &= x + \mu P_{n-1} y \ (\because (4.8)(4.20)) \end{aligned} \tag{4.22}$$

また,$r^t P_{n-1} v + \mu = b_n$ より,

$$\begin{aligned} \mu &= b_n - r^t P_{n-1} v \\ &= b_n - r^t P_{n-1} x - \mu r^t y \\ &= (b_n - r^t P_{n-1} x)/(1 + r^t y) \end{aligned} \tag{4.23}$$

と表すことができる. 要するに,$k = 1 : n$ の間で

- $T_k x^{(k)} = b^{(k)} = (b_1, \ldots, b_k)^t$
- $T_k y^{(k)} = -r^{(k)} = (r_1, \ldots, r_k)^t$

を平行して解いていくことによって, 能率的に $T_n x^{(n)} = b^{(n)}$ を解くことができるということである. これを次のアルゴリズムとしてまとめることができる.

Levinson-Durbin アルゴリズム　$b, r \in \mathbb{F}^n$ に対して,Yule-Walker 方程式 $Tx = b, Ty = -r$ の近似解 $x, y \in \mathbb{F}^n$ を求めるアルゴリズム.

function $[x, y]$=Solve_Toeplitz(b, r)

$y(1) = -r(1); x(1) = b(1); \beta = 1; \alpha = -r(1)$

for　$k = 1 : n-1$

$\beta = (1 - \alpha^2)\beta$

$\mu = (b(k+1) - r(1:k)^t x(k:-1:1))/\beta$

$v(1:k) = x(1:k) + \mu y(k:-1:1)$

$$x(1:k+1) = \begin{bmatrix} v(1:k) \\ \mu \end{bmatrix}$$

if　$k < n-1$

$\alpha = (-r(k+1) + r(1:k)^t y(k:-1:1))/\beta$

$z(1:k) = y(1:k) + \alpha y(k:-1:1)$

$$y(1:k+1) = \begin{bmatrix} z(1:k) \\ \alpha \end{bmatrix}$$

end

end

このアルゴリズムは $4n^2 flops$ の計算量で実行できる.

4.3.2 アルゴリズムの適用

さてここで, 要素として $\{r_1, \ldots, r_{n-1}\}, \{p_1, \ldots, p_{n-1}\}, \{b_1, \ldots, b_n\}$ が与えられていて, 次のような形の線型方程式を解くとする.

$$\begin{bmatrix} 1 & r_1 & r_2 & \cdots & r_{n-1} \\ p_1 & 1 & r_1 & \cdots & r_{n-2} \\ p_2 & p_1 & 1 & \ddots & r_{n-3} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ p_{n-1} & p_{n-2} & p_{n-3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} \tag{4.24}$$

しかし, これは次の 3 式のそれぞれに解法を持っていれば容易に解くことができる. なお,$w, p \in \mathbb{F}^{n-1}$ とする.

$$T_{n-1}^t y = -r \tag{4.25}$$

$$T_{n-1} w = -p \tag{4.26}$$

$$T_{n-1} x = b \tag{4.27}$$

ここで, 非対称 Toeplitz 行列の構造的性質から, 次の 2 つの関係式が導かれる.

$$[T_n^t]^{-1} = [T_n^{-1}]^t \tag{4.28}$$

$$T_n P_n = P_n T_n^t \tag{4.29}$$

以下では上式の関係を用いて,Levinson-Durbin アルゴリズムの適用を試みる.

- $T_n^t y^{(n)} = -r^{(n)}$

$$\begin{bmatrix} T_{n-1}^t & P_{n-1} p \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} z \\ \alpha \end{bmatrix} = - \begin{bmatrix} r \\ r_n \end{bmatrix} \tag{4.30}$$

上式より以下の 2 式が導かれる.

$$\begin{aligned} z &= [T_{n-1}^t]^{-1}(-r - \alpha P_{n-1}p^t) \\ &= -[T_{n-1}^t]^{-1}r - \alpha[T_{n-1}^t]^{-1}P_{n-1}p^t \\ &= -[T_{n-1}^t]^{-1}r - \alpha P_{n-1}T_{n-1}^{-1}p^t \\ &= y + \alpha P_{n-1}w \ (\because (4.25)(4.26)) \end{aligned} \tag{4.31}$$

$$\begin{aligned} \alpha &= -r_n - r^t P_{n-1}z \\ &= -r_n - r^t P_{n-1}y - \alpha r^t w \ (\because (4.31)) \\ &= -(r_n + r^t P_{n-1}y)/(1 + r^t w) \end{aligned} \tag{4.32}$$

$T_n w^{(n)} = -p^{(n)}, T_n x^{(n)} = b^{(n)}$ についても同様に関係式が導かれる. なお,$u \in \mathbb{F}^{n-1}, \nu \in \mathbb{F}$ とする.

- $T_n w^{(n)} = -p^{(n)}$

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ p^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} u \\ \nu \end{bmatrix} = - \begin{bmatrix} p \\ p_n \end{bmatrix} \tag{4.33}$$

上式より以下の 2 式が導かれる.

$$\begin{aligned} u &= T_{n-1}^{-1}(-p - \nu P_{n-1}r) \\ &= -T_{n-1}^{-1}p - \nu T_{n-1}^{-1}P_{n-1}r \\ &= -T_{n-1}^{-1}p - \nu P_{n-1}[T_{n-1}^t]^{-1}r \\ &= w + \nu P_{n-1}y \ (\because (4.25)(4.26)) \end{aligned} \tag{4.34}$$

$$\begin{aligned} \nu &= -p_n - p^t P_{n-1}u \\ &= -p_{n-1} - p^t P_{n-1}w - \nu p^t y \ (\because (4.34)) \\ &= -(p_n + p^t P_{n-1}w)/(1 + p^t y) \end{aligned} \tag{4.35}$$

- $T_n x^{(n)} = b^{(n)}$

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ r^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} v \\ \mu \end{bmatrix} = \begin{bmatrix} b \\ b_n \end{bmatrix} \tag{4.36}$$

上式より以下の 2 式が導かれる.

$$\begin{aligned} v &= T_{n-1}^{-1}(b - \mu P_{n-1}r) \\ &= T_{n-1}^{-1}b - \mu T_{n-1}^{-1}P_{n-1}r \\ &= T_{n-1}^{-1}b - \mu P_{n-1}[T_{n-1}^t]^{-1}r \\ &= x + \mu P_{n-1}y \ (\because (4.25)(4.27)) \end{aligned} \tag{4.37}$$

$$\begin{aligned} \mu &= b_n - p^t P_{n-1}v \\ &= b_n - p^t P_{n-1}x - \mu p^t y \ (\because (4.37)) \\ &= (b_n - p^t P_{n-1}x)/(1 + p^t y) \end{aligned} \tag{4.38}$$

4.3.3 式の変形

以下では式の変形について触れる. 非対称な場合では 3 つの式が相互に関係してくるので, まとめて取り上げることにする.

式 (4.32)(4.35)(4.38) にある,$1 + r^t w, 1 + p^t y$ を以下のように簡略化させる.

$$\begin{aligned} \alpha'_k &= 1 + [r^{(k)}]^t w^{(k)} \\ &= 1 + \begin{bmatrix} [r^{(k-1)}]^t & r_k \end{bmatrix} \begin{bmatrix} w^{(k-1)} + \beta_{k-1}P_{k-1}y^{(k-1)} \\ \beta_{k-1} \end{bmatrix} \\ &= 1 + [r^{(k-1)}]^t w^{(k-1)} + \beta_{k-1}([r^{(k-1)}]^t P_{k-1}y^{(k-1)} + r_k) \\ &= \alpha'_{k-1} + \beta_{k-1}(-\alpha_{k-1}\alpha'_{k-1}) \\ &= (1 - \beta_{k-1}\alpha_{k-1})\alpha'_{k-1} \end{aligned} \tag{4.39}$$

$$
\begin{aligned}
\beta'_k &= 1 + [p^{(k)}]^t y^{(k)} \\
&= 1 + \left[\begin{array}{cc} [p^{(k-1)}]^t & p_k \end{array}\right] \left[\begin{array}{c} y^{(k-1)} + \alpha_{k-1} E_{k-1} w^{(k-1)} \\ \\ \alpha_{k-1} \end{array}\right] \\
&= 1 + [p^{(k-1)}]^t y^{(k-1)} + \alpha_{k-1}([p^{(k-1)}]^t E_{k-1} w^{(k-1)} + p_k) \\
&= \beta'_{k-1} + \alpha_{k-1}(-\beta_{k-1}\beta'_{k-1}) \\
&= (1 - \alpha_{k-1}\beta_{k-1})\beta'_{k-1} \qquad (4.40)
\end{aligned}
$$

よって, 以下の式より, 各々の解を逐次的に求めることができる.

- $T_n^t y^{(n)} = -r^{(n)}$

$$
\begin{aligned}
z &= y + \alpha P_{n-1} w \qquad (4.41) \\
\alpha &= -(r_n + r^t P_{n-1} y)/\alpha'_{n-1} \qquad (4.42) \\
\alpha'_k &= (1 - \beta_{k-1}\alpha_{k-1})\alpha'_{k-1} \qquad (4.43)
\end{aligned}
$$

- $T_n w^{(n)} = -p^{(n)}$

$$
\begin{aligned}
u &= w + \beta P_{n-1} y \qquad (4.44) \\
\beta &= -(p_n + p^t P_{n-1} w)/\beta'_{n-1} \qquad (4.45) \\
\beta'_k &= (1 - \alpha_{k-1}\beta_{k-1})\beta'_{k-1} \qquad (4.46)
\end{aligned}
$$

- $T_n x^{(n)} = b^{(n)}$

$$
\begin{aligned}
v &= x + \mu P_{n-1} y \qquad (4.47) \\
\mu &= (b_n + p^t P_{n-1} x)/\beta'_{n-1} \qquad (4.48) \\
\beta'_k &= (1 - \alpha_{k-1}\beta_{k-1})\beta'_{k-1} \qquad (4.49)
\end{aligned}
$$

4.3.4 非対称 Toeplitz 行列の逆行列の導出

次に非対称の場合での逆行列の導出について触れる. いま, 非対称 Toeplitz 行列の逆行列 T_n に対する近似逆行列 R_n を,

$$R_n = \begin{bmatrix} B_{n-1} & e \\ f^t & g \end{bmatrix} \tag{4.50}$$

とするとき,$T_n R_n = I_n$ から以下の式が成り立つ. ここで,$e, f \in \mathbb{F}^{n-1}, g \in \mathbb{F}$ とする.

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ p^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} B_{n-1} & e \\ f^t & g \end{bmatrix} = I_n \tag{4.51}$$

ここで

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ p^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} e \\ g \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.52}$$

より, 以下の式を得る.

$$\begin{aligned} e &= -gT_{n-1}^{-1}P_{n-1}r \\ &= -gP_{n-1}[T_{n-1}^t]^{-1}r \\ &= gP_{n-1}y \ (\because (4.8)) \end{aligned} \tag{4.53}$$

$$\begin{aligned} g &= 1 - p^t P_{n-1}e \\ &= 1 - gp^t y \ (\because (4.53)) \\ &= 1/(1 + p^t y) \end{aligned} \tag{4.54}$$

また,

$$\begin{bmatrix} T_{n-1} & P_{n-1}r \\ p^t P_{n-1} & 1 \end{bmatrix} \begin{bmatrix} B_{n-1} \\ f^t \end{bmatrix} = \begin{bmatrix} I_{n-1} \\ 0 \end{bmatrix} \tag{4.55}$$

より, 以下の式を得る.

$$\begin{aligned} B_{n-1} &= T_{n-1}^{-1}(I_{n-1} - P_{n-1} r f^t) \\ &= T_{n-1}^{-1} - T_{n-1}^{-1} P_{n-1} r f^t \\ &= T_{n-1}^{-1} - P_{n-1}[T_{n-1}^t]^{-1} r f^t \\ &= T_{n-1}^{-1} + P_{n-1} y f^t \ (\because (4.8)) \end{aligned} \tag{4.56}$$

$$\begin{aligned} f^t &= -p^t P_{n-1} B_{n-1} \\ &= -p^t P_{n-1}(T_{n-1}^{-1} + P_{n-1} y f^t) \ (\because (4.56)) \\ &= -p^t P_{n-1} T_{n-1}^{-1} - p^t y f^t \\ &= -p^t [T_{n-1}^t]^{-1} P_{n-1} - p^t y f^t \\ &= -[T_{n-1}^t]^{-1} p^t P_{n-1} - p^t y f^t \\ &= w P_{n-1} - p^t y f^t \ (\because (4.26)) \\ &= w P_{n-1}/(1 + p^t y) \end{aligned} \tag{4.57}$$

これにより, 非対称 Toeplitz 行列の時も, 逆行列を求めることができた.

これより, 近似逆行列 R_n を求めるアルゴリズムは次のように表される.

Levinson-Durbin アルゴリズムにより逆行列を求めるアルゴリズム $r, p \in \mathbb{F}^n$ に対して非対称 $Toeplitz$ 行列 T_n に対する近似逆行列 $R_n \in \mathbb{F}^{n\times n}$ を求めるアルゴリズム.

function R_n =Toeplitz_Inverse(r, p)

$[t, w, x] = u\_solve(r, p, b)$

$g = 1/(1 + p(1 : n-1)^t y(1 : n-1))$

$e(1 : n-1) = gy(n-1 : -1 : 1)$

$f(1 : n-1) = g\omega(n-1 : -1 : 1)$

$B(1, 1) = g$

$B(1, 2 : n) = e(n-1 : -1 : 1)^t$

$k = 0$

for $\quad i = 2 : n-1$

$\quad k = k + n + 1 - i$

$\quad l = n + 1 - i$

$\quad$ for $\quad j = 2 : n + 1 - i$

$\quad\quad B(k, j+1) = B(k, j-l-1)$

$\quad\quad\quad\quad +y(j-1)f(n+1-i) - y(n+1-i)f(j-1)$

$\quad$ end

end

このアルゴリズムは $6n^2 flops$ で計算できる. またこのアルゴリズムでは, 逆行列の persymmetric 性を利用し, 行列全体の $1/2$ の成分を計算する.

4.4 必要とするメモリ量を削減する精度保証法

Toeplitz 行列は $O(n)$ のメモリ量があれば格納できるが, 精度保証に必要な近似逆行列を格納するには $O(n^2)$ のメモリ量が必要である. そのために近似解を出すことができる計算機環境において, 近似逆行列を格納するメモリが不足するために精度保証ができない可能性がある. そこで, 近似逆行列を用いて計算する部分に必要なメモリ量を $O(n)$ に抑える工夫をする.

精度保証において近似逆行列を用いて計算する部分は, 近似逆行列が行単位で求まっていれば精度保証の計算をすることが可能である. ここで近似逆行列の導出過程に着目すると逆行列を行単位で求めることができ, また $i+1$ 行目の逆行列の成分は i 行目の逆行列の成分から求めることが可能である. よって, 近似逆行列の i 行目を求めて精度保証の計算を行い, 計算を終えると近似逆行列の $i+1$ 行目を求め i 行目を格納しているベクトルに上書きすることによりメモリを $O(n)$ に抑えることが可能である.

アルゴリズムの説明上の煩雑さを避けるため, 以下の関数を定義する.Toeplitz 行列の要素から第 i 列を返す関数を Toeplitz 行列が対称であれば $T = Col\_T(r, i)$, 非対称であれば $T = Col\_T(r, p, i)$ とし, 逆行列の i 行目のベクトルを q としたときに $i+1$ 行目の逆行列の成分を表すベクトルを求める関数を $Row\_Inv(q, i)$ とする. また $\beta/(1-\alpha)$ の上限 res を求める関数を $res = Distinction(\alpha, \beta)$ とし, 残差 $T_n\widetilde{x} - b$ を計算する関数を Toeplitz 行列が対称であれば $residual(r, \widetilde{x}, b)$, 非対称であれば $residual(r, p, \widetilde{x}, b)$ とする. これらの関数を用いて, 本節と前節で提案した手法を適応した精度保証アルゴリズムを以下に示す. アルゴリズム中では $B_nT_n - I_n$ の最大値ノルムは $B_nT_n - I_n$ の成分を列単位で求め, ベクトルの絶対値和を求めることで評価をした.

Algorithm 1 function err =OurMethod1$(r, \widetilde{x}, b)$

$T = zeros(N,1); Tinv = zeros(N,1); \quad G = zeros(N,1); \quad Rr = zeros(N,1);$

$y = Solve\_Toeplitz(r);$

$g = 1/(1 + r' * y); \quad v(1; N-1) = g * y(N-1 : -1 : 1);$

$Tinv(1) = g; \quad Tinv(2 : N) = v(N-1 : 1);$

$rd = residual(r, \widetilde{x}, b);$

for $i = 1 : N$

$T = Col\_T(r, 1);$

$G(1) = G(1) + abs(Tinv * T);$

end

$Rr(1) = Tinv(1 : N) * rd; \quad Rr(N-1) = Tinv(N : -1 : 1) * rd;$

for $i = 2 : floor(N/2) + 1$

$Tinv(1 : N) = Row\_Inv(Tinv, i);$

$Rr(i) = Tinv(1 : N)' * rd; \quad Rr(N-i) = Tinv(N : -1 : 1)' * rd;$

for $j = 1; N$

$T = Col\_T(r, j)$

$G(i) = G(i) + abs(Tinv * T);$

end

end

$G\_norm = norm(G, Inf); \quad Rr\_norm = norm(Rr, Inf);$

$err = Distinction(G\_norm, Rr\_norm);$

近似逆行列は本来であれば対称 Toeplitz 行列の逆行列は全成分の 1/4 を計算すればよいが, メモリを削減するために行単位で計算するため全成分の 1/2 を計算する

ことになる.次に非対称 Toeplitz 行列の場合のアルゴリズムを以下に記す.

Algorithm 2 function err =OurMethod2(r, p, b)

$T = zeros(2 * N - 1, 1); Tinv = zeros(N, 1);$ $G = zeros(N, 1);$ $Rr = zeros(N, 1);$

$[t, y, w] = Solve\_Toeplitz(r, p);$

$g = 1/(1 + p'y);$ $e(1 : N - 1) = gy(N - 1 : -1 : 1);$

$f'(1 : N - 1) = gw'(N - 1 : -1 : 1);$ $Tinv(1) = g;$ $Tinv(2 : N) = e(N - 1 : 1)';$

$rd = residual(r, p, \widetilde{x}, b);$

for $i = 1 : N$

$T = Col\_T(r, p, 1);$

$G(1) = G(1) + abs(Tinv * T);$

end

$Rr(1) = Tinv(1 : N) * rd;$

for $i = 2 : N$

$Tinv(1 : N) = Row\_Inv(Tinv, i);$

$Rr(i) = Tinv(1 : N)' * rd;$

for $j = 1; N$

$T = Col\_T(r, p, j)$

$G(i) = G(i) + abs(Tinv * T);$

end

end

$G\_norm = norm(G, Inf); Rr\_norm = norm(Rr, Inf);$

$err = Distinction(G\_norm, R\_norm);$

またアルゴリズム 1,2 を丸めモードの変更や区間演算を用いて計算することにより誤差上限が計算できる.

4.5 一般最適化フィルタ

ある定常な確率過程からの観測を $x(n)$ で表す. これは, 直接は観測できない他の定常確率過程からの実現値 $d(n)$ と何らかの関係があるものとする. 例えば, 何らかの信号源から生成される信号が $d(n)$ であるが, それが伝搬する途中で歪んだり誤差が加わって $x(n)$ として観測された場合などに相当する. ここでの信号処理の目的は, 観測系列 $x(n)$ から $d(n)$ を推定することである. 特に, 現在の観測 $x(n)$ と過去の $P-1$ 個の観測 $x(n-1), x(n-2), \cdots, x(n-P+1)$ だけを使って $d(n)$ を推定することを考える.$d(n)$ は「所望」系列などと呼ばれることが多い. この種の問題を表 (4.5) に示した. 表中の $s(n)$ は処理対象の信号成分を表し,$\eta(n)$ は雑音成分である.

表 4.1: Wiener Filter

問題	式表現	所望信号
雑音中の信号のフィルタリング	$x(n) = s(n) + \eta(n)$	$d(n) = s(n)$
雑音中の信号の予測	$x(n) = s(n) + \eta(n)$	$d(n) = s(n+p); p > 0$
雑音中の信号の平滑化	$x(n) = s(n) + \eta(n)$	$d(n) = s(n-q); q > 0$
線形予測	$x(n) = s(n-1)$	$d(n) = s(n)$
一般非線形型問題	$x(n) = G(s(n), \eta(n))$	$d(n) = s(n)$

4.5.1 離散 Wiener-Hopf 方程式

まず,$x(n)$ と過去の $P-1$ 個の観測から, 何らかの線形フィルタで予測される所望信号推定値を $\hat{d}(n)$ と書くことにする.

$$\hat{d}(n) = \sum_{k=n-P+1}^{n} h^*(n,k)x(k) = \sum_{l=0}^{P-1} h^*(n,n-l)x(n-l) \tag{4.58}$$

ここでの問題は, 次式で示す推定誤差 ϵ の二乗平均（分散）を最小とする線形フィルタのインパルス応答 $h(n,k)$ を決定することである.

$$\epsilon(n) = d(n) - \hat{d}(n) \tag{4.59}$$

この問題は「線形最小二乗推定」問題であるから, 直行性原理が適用できる. すなわち, 任意の時刻での観測値 $x(n-i)$ と推定誤差が直行すればよいので,

$$E\left\{x(n-i)\epsilon^*(n)\right\} = E\left\{x(n-i)\left(d^*(n) - \sum_{l=0}^{P-1} h(n,n-l)x^*(n-l)\right)\right\} = 0 \tag{4.60}$$

となる. 上式は, 観測値ベクタの自己相関関数 $R_x = E\left\{xx^{*T}\right\}$ および所望信号と観測値ベクタの相互相関ベクタ $r_{dx} = E\left\{d^*(n)x\right\}$ の要素を用いて次のように書くことができる.

$$r_{dx}(n,n-i) = \sum_{l=0}^{P-1} R_x(n-i,n-l)h(n,n-l); \quad i = 0,1,\cdots,P-1 \tag{4.61}$$

この式は Wiener-Hopf 方程式の離散表現となっており,R_x と r_{dx} がわかれば, フィルタのインパルス応答を求めることができる. 一方, 平均二乗誤差の最小値は, 直交化原理に基づいて次式で計算できる（$R_d(n,n)$ は所望信号 $d(n)$ の自己相関関数を表す）.

$$\begin{aligned}
\sigma^2(n) &= E\left\{d(n)\epsilon^*(n)\right\} \\
&= E\left\{d(n)\left(d^*(n) - \sum_{l=0}^{P-1} h(n,n-l)x^*(n-l)\right)\right\} \\
&= R_d(n,n) - \sum_{l=0}^{P-1} h(n,n-l)r_{dx}^*(n,n-l)
\end{aligned} \tag{4.62}$$

ここで, シフト不変性が成り立つ場合にはもう少し簡単に定式化できて, 次のように書ける（所望信号の分散を σ_d^2 とした）. また, 最小誤差分散の式の変形には $R_x^{*T} = R_x$ を使用している.

$$r_{dx}(i) = \sum_{l=0}^{P-1} R_x(l-i)h(l);\ i = 0, 1, \cdots, P-1 \ \rightarrow\ r_{dx} = R_x h \tag{4.63}$$

$$\sigma_x^2 = \sigma_d^2 - \sum_{l=0}^{P-1} h(l) r_{dx}^*(l) = \sigma_d^2 - h^{*t} r_{dx} = \sigma_d^2 - r^{*t} R_x^{-1} r_{dx} \tag{4.64}$$

ここで,$P = N$ の場合について考えてみると, 式 (4.63) および式 (4.64) は具体的に次のように書ける.

$$\begin{bmatrix} R_x(0) & R_x(1) & R_x(2) & \cdots & R_x(N-1) \\ R_x(-1) & R_x(0) & R_x(1) & \ddots & \vdots \\ R_x(-2) & R_x(-1) & R_x(0) & \ddots & R_x(2) \\ \vdots & \ddots & \ddots & \ddots & R_x(1) \\ R_x(-N+1) & \cdots & R_x(-2) & R_x(-1) & R_x(0) \end{bmatrix} \begin{bmatrix} h^*(0) \\ h^*(1) \\ h^*(2) \\ \vdots \\ h^*(N-1) \end{bmatrix} = \begin{bmatrix} r_{dx}^*(0) \\ r_{dx}^*(1) \\ r_{dx}^*(2) \\ \vdots \\ r_{dx}^*(N-1) \end{bmatrix} \tag{4.65}$$

$$\sigma_\epsilon^2 = R_d(0) - \sum_{k=0}^{N-1} \{h^*(k) r_{dx}(k)\} \tag{4.66}$$

4.5.2 雑音除去フィルタリング

今までの議論を元に, ディジタル信号処理における雑音除去フィルタリングを実際に行ってみることにする. ある定常信号 $s(n)$ に雑音 $\eta(n)$ が加わって $x(n)$ が観測されたとすると,$s(n)$ の自己相関関数は $R_s(l) = 2 \times (0.8)^{|l|}$ と知られており,$\eta(n)$ の自己相関関数も $R_\eta(l) = 2\delta(l)$ であることがわかっているものとする.

$$\delta(l) = \begin{cases} 1 & (l = 0) \\ 0 & (l \neq 0) \end{cases} \tag{4.67}$$

ここで, 現在の時刻 n における信号の値 $s(n)$ を所望信号 $d(n)$ とする場合（雑音 $\eta(n)$ の除去を行うフィルタリングの場合）について考える. 信号 $s(n)$ と雑音 $\eta(n)$ が無

相関だとすれば次の関係が成り立つ.

$$
\begin{aligned}
R_d(l) &= R_s(l) = 2(0.8)^{|l|} && (4.68)\\
r_{dx}(l) &= E\left\{s(n)x(n-l)\right\} = E\left\{s(n)\left\{s(n-l)+\eta(n-l)\right\}\right\} \\
&= R_s(l) = 2(0.8)^{|l|} && (4.69)\\
R_x(l) &= E\left\{x(n)x(n-l)\right\} = E\left\{\left\{s(n)+\eta(n)\right\}\left\{s(n-l)+\eta(n-l)\right\}\right\} \\
&= R_s(l) + R_\eta(l) = 2(0.8)^{|l|} + 2\delta(l) && (4.70)
\end{aligned}
$$

これを元に式 (4.65) を Levinson-Durbin アルゴリズムを使用して解くことにより, インパルス応答 $h(0), h(1), \cdots, h(N-1)$ を得ることができ, このインパルス応答を使った平均二乗誤差の最小値の上限を, 式 (4.66) に代入することにより求めることができる.

第5章　高速精度保証の実装と結果

5.1　実装方法

逆行列 R の導出,RA の計算を前章の方法により行い, 実際に精度保証付き数値計算を行ってみる. 連立一次方程式 $Ax = b$ に対して,A の近似逆行列 R を求め, 近似解 $x = Rb$ を計算する. このとき, 真の解の存在の十分条件を検証し, もし存在する場合には, 真の解 x^* と x との間の誤差 $err = \| x - x^* \|_\infty$ を計算するプログラムを作成した. またそれを,LU 分解法を用いて通常通り計算したものと比較してみた. 実装には C 言語と数値計算ツール Matlab6.5 を使用した. ただし計算は MATLAB 上から C 言語の $mexfunction$ を呼び出す形で高速化を図った.

また, 実用例として取り上げた「雑音除去フィルタリング」においても, 前述のアルゴリズムを用いて実装を行った. この際, サンプリングするデータの間隔を狭めることにより, フィルタリングの精度が高くなるかということも検証した. その方法として, 式 (4.68),(4.69) 中の「$2(0.8)^{|l|}$」を「$2(0.8)^{\frac{|l|}{m}}$」とすることにより, 今までのデータの間隔を m 分割している.

5.2 実行結果

上記のプログラムを実装した結果を以下に示す. なお、実行環境は以下のとおりとする.

- CPU ··· PentiumM 1.10GB
- Main Memory ··· 256MB
- OS ··· WindowsXP
- Compiler ··· Visual C 6.0

表中の n は次元数を,$time1$ は近似解を得るのに要した時間を表し,$time2$ は精度保証に要した時間を表す. また err は近似解に対する誤差限界を表し, また「 - 」はメモリ不足により測定不能であったことを示す.

表 5.1: LU 分解法による精度保証

n	$time1[s]$	$time2[s]$	err	$\|RA-I\|_\infty$
500	0.008	1.029	2.149E-07	2.148E-10
1000	0.020	7.248	6.877E-06	2.519E-09
2000	0.118	80.514	1.806E-05	8.013E-09
3000	0.242	783.431	3.491E-04	2.916E-08
5000	0.380	-	-	-

表 5.2: 対称 Toeplitz 行列による精度保証

n	$time1[s]$	$time2[s]$	err	$\|RA-I\|_\infty$
500	0.008	0.671	4.348E-09	7.436E-08
1000	0.020	3.936	1.401E-07	1.584E-06
2000	0.118	30.544	2.795E-07	8.253E-06
3000	0.242	104.262	1.662E-06	6.836E-05
5000	0.380	479.083	3.286E-06	2.244E-04
10000	1.011	3828.556	3.166E-05	1.878E-03

表 5.3: 非対称 Toeplitz 行列による精度保証

n	$time1[s]$	$time2[s]$	err	$\|RA-I\|_\infty$
500	0.006	1.135	4.596E-10	1.633E-08
1000	0.036	7.691	1.305E-09	1.229E-07
2000	0.134	60.259	1.144E-08	2.213E-06
3000	0.264	209.113	4.576E-08	3.341E-06
5000	0.509	964.206	3.295E-07	8.754E-06
10000	1.011	7685.741	1.234E-06	1.002E-04

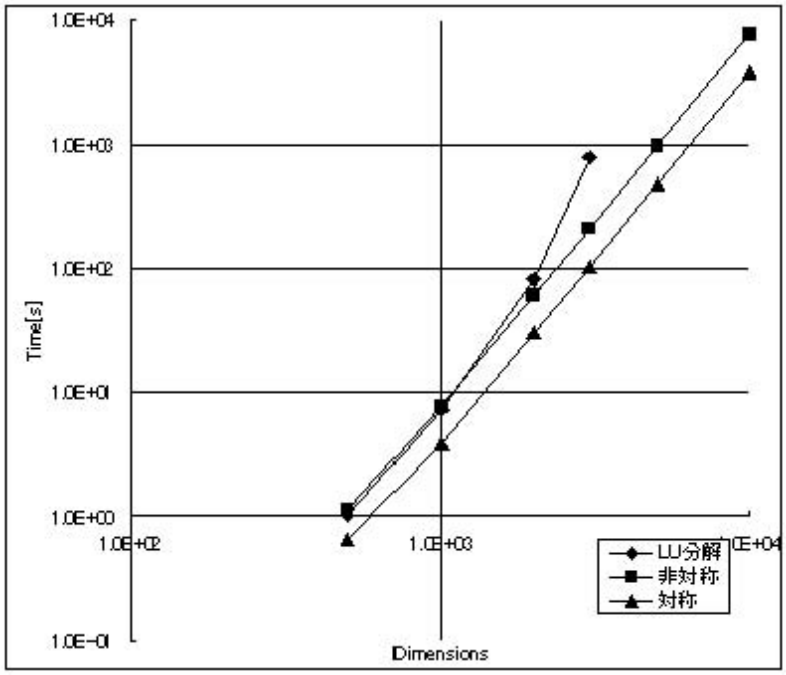


図 5.1: 次元数と精度保証にかかる時間の関係

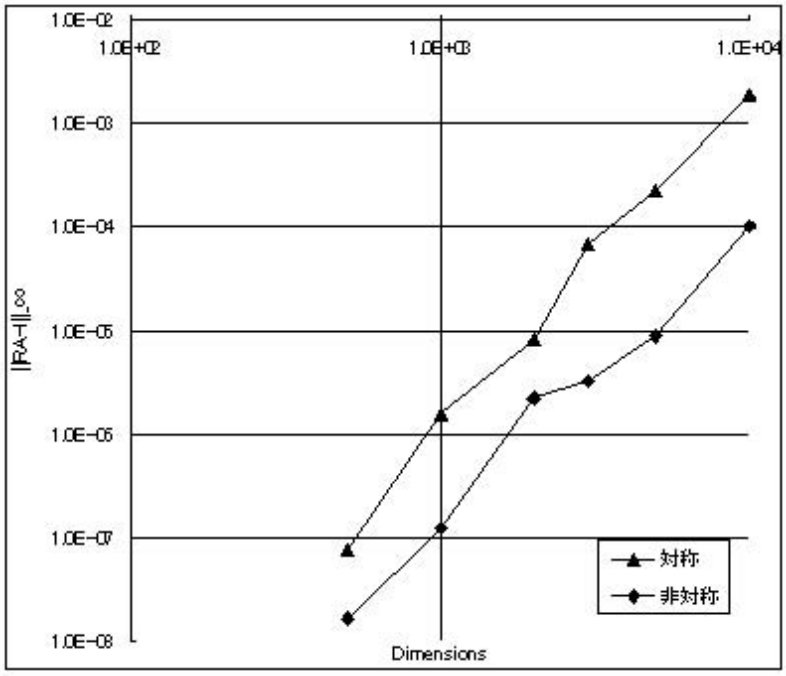


図 5.2: 次元数と $\|RA - I\|_\infty$ の関係

表 5.4: 雑音除去フィルタリングにおける平均二乗誤差

n	$time1[s]$	$time2[s]$	err	σ_ϵ^2	$\|RA-I\|_\infty$
500	0.015	0.541	1.745E-14	0.7500	7.759E-13
1000	0.020	3.795	4.854E-14	0.7500	1.587E-12
2000	0.070	29.766	1.089E-13	0.7500	3.211E-12
3000	0.173	103.398	1.701E-13	0.7500	4.834E-12
5000	4.236	478.848	2.313E-13	0.7500	7.845E-12

表 5.5: 雑音除去フィルタリングにおける平均二乗誤差 ($n = 1000$ とし分割数 m のみを変更)

m	$time1[s]$	$time2[s]$	err	σ_ϵ^2	$\|RA-I\|_\infty$
1	0.017	3.795	4.854E-14	7.500E-01	1.587E-12
2^8	0.020	3.792	1.080E-14	8.013E-02	6.982E-11
2^{16}	0.023	3.816	3.973E-15	5.262E-03	1.165E-10
2^{32}	0.020	3.802	2.538E-15	1.998E-03	1.420E-10
2^{64}	0.020	3.782	1.781E-16	1.998E-03	1.409E-10

第6章　結び

6.1　まとめ

第 2 章では, 精度保証付き数値計算を論じる上で欠かせない浮動小数点や区間演算などの概念について述べ,Toeplitz 行列が実際にどのようなアプリケーションに現れているかということに触れた.

第 3 章では, 連立一次方程式の近似解の求め方として LU 分解法について述べた. また, その解の精度保証をするための定理を挙げて研究した結果, 処理を高速化するために係数行列 A の近似逆行列 R とその積の RA の演算方法を改善させればよいということに帰着した.

第 4 章では, 係数行列, つまり Toeplitz 行列の近似逆行列を求めるための, 二つのアルゴリズムの概要を示した. そのアルゴリズムを利用することによって, 近似逆行列を求める演算量が $O(n^2)$ で済むということがわかった. さらに, 新たなアルゴリズムを提示し, 対称 Toeplitz 行列, 非対称 Toeplitz 行列に関わらず同じ議論ができるということを示した.

第 5 章では, 第 4 章で述べたアルゴリズムによって精度保証付き数値計算を実装するプログラムを示し, その結果として実行時間と近似解と真の解の誤差を表した. また, 実用例として「雑音除去フィルタリング」を取り上げ, その効果の程を試した.

6.2 実行結果の検証

本論文ではToeplitz行列を係数行列に持つ大規模な連立一次方程式の精度保証の方法を提案した. 一般的に使用されているLU分解法による逆行列の導出の場合に$O(n^3)$の計算量が必要なのに対して, 提案手法は$O(n^2)$の計算量で実行でき,Toeplitz行列の格納に必要なメモリ量を$O(n)$に抑えて精度保証が行えることを示した. この提案手法により, 大規模な連立一次方程式の数値解の精度保証が可能になった.

提案手法を実際に実装して精度保証をし, 実行時間を両対数グラフにて検証した. その結果, 表(5.2),(5.3)より, 本論文に使用した計算機環境では5000次元の近似逆行列の成分を全てメモリに保持することは不可能であるが, 提案手法のように段階的に逆行列の成分を計算し利用・破棄を繰り返す方式によりメモリ量の問題を解決し, 大規模な連立一次方程式でも精度保証することが可能になった.

図(5.1)を見ると, 次元数が低いうちは, 非対称Toeplitz行列よりもLU分解法の方が若干ではあるが速度が速くなっている. これは,MATLABという数値計算ツールが行列計算に最適化されているために起きた現象であると考えられる. しかし次元数を上げた場合, 提案手法では, 以前よりもメモリ領域の空きが大きくなることや, メモリアクセスが少なくなることなどから, ページアウトやスワッピングを起こしにくく, 実行時間の増大を抑えつつも, 少ないメモリ環境でも扱える次元数を上昇させることができるという結果に繋がった.

また, 次元数と$\|RA-I\|_\infty$との関係を表(5.2)のように両対数グラフを書いてみた. ここで$\|RA-I\|_\infty < 1$を満たすときに精度保証ができることに注目して, 最小二乗法によりグラフの近似直線を作り, それをもとに$\|RA-I\|_\infty = 1$となる次元数を計算してみた. その結果, 対称Toeplitz行列の場合は$n \cong 6.097 \cdot 10^4$となったので約6万次元, 非対称Toeplitz行列の場合は$n \cong 2.572 \cdot 10^5$となったので約25万次元まで精度保証ができることが推測できた.

「雑音除去フィルタリング」の実験では, 表 (5.4) のように今回使用したサンプルでは次元数を上げる, つまりサンプリングデータ数を増やしてもあまり精度の上昇は見られなかった. しかし, 表 (5.5) のように分割数を増やし取得データの間隔を狭めることにより, フィルタリングの精度を上昇させられることが確認できた. また Toeplitz 行列の性質を用いたことで, 大規模な問題にも対応できることが示された. ここで, 分割数を 2^{32} にした時と 2^{64} にした時とで σ_ϵ^2 や $\|RA - I\|_\infty$ の値に変化が見られなくなったが, これは分割数を大きくしすぎたため, サンプリングデータをグラフにするとどちらも同じような形（直線）になるので, それ以上の計算精度が得られなくなってしまったものと考えられる.

6.3 今後の課題

節 (6.2) にて検証した通り,Toeplitz 行列を扱う上で 6 万次元程度の行列までは精度保証ができることが推測された. ただ, 現時点ではそのような大規模行列の精度保証をするために莫大な時間がかかり, 精度保証の高速化が望まれるところである. この精度保証を行うための計算量を $O(n^3)$ から $O(n^2)$ に落として, 大規模連立一次方程式の近似解に対して, 数倍程度の手間で実行できるようにすることが目標であり今後の課題となる. これにより, さらに大規模な連立一次方程式の精度保証を少ない実行時間で行うことができるであろう.

謝辞

本研究を進めるに当たり, 終始丁寧な御指導及び御激励を賜り, その他多くの面でも色々と御面倒を見て下さり御助言を与えて下さいました **大石 進一 教授** に深く感謝いたします.

また, 終始丁寧なご指導と御教示をして下さいました,21 世紀 COE 客員講師, 丸山 晃佐 氏, 中谷 祐介 氏, JST 研究員, 荻田 武史 氏, 客員講師, 松永 奈美 氏に大いに感謝いたします.

また, 研究指導や日常生活において色々とお世話になりました, 早稲田大学大学院大石研究室博士課程 2 年 尾崎 克久 氏 に大いに感謝いたします.

また, 同じ学年のメンバーとしていろいろな意見の交換や協力などをして下さいました, 大石研究室修士課程 2 年 坂内 太郎 氏, 島本 誠 氏, 鈴木 大育 氏, 関本 竜平 氏, 徳永 克久 氏, 細田 幸裕 氏, 横山 大輔 氏 に深く感謝いたします.

最後に, 研究だけでなく日常の生活の中でお世話になりました大石研究室の皆様に深く感謝いたします.

参考文献

[1] Golub and Van Loan(1993). *Matrix Computations*, Johns Hopkins University Press, 193-205.

[2] 大石進一, 数値計算, 裳華房,(2001-4).

[3] 大石進一, 精度保証付き数値計算, コロナ社,(2000-1).

[4] 大石進一,Linux 数値計算ツール, コロナ社,(2001-6).

[5] 大石進一,MATLAB による数値計算, 培風館,(2001-7).

[6] 小林一行,MATLAB 活用ブック, 秀和システム,(2001-7).

[7] ANSI/IEEE: *IEEE Standard for Binary Floating Point Arithmetic*, Std 754–1985 edition, IEEE, New York, 1985.

[8] S. Oishi, S.M. Rump, "Fast Verification of Solutions of Matrix Equations," Numer. Math., vol.90, no.4, pp.755–773, 2002.

[9] T. Sunaga: Theory of an interval algebra and its application to numerical analysis, RAAG Memoirs, **2** (1958), 547–564.

[10] G.H. Golub, C.F. Van Loan, Matrix Computations, 3rd edition, Johns Hopkins University Press, Baltimore and London, 1996.

[11] S. M. Rump, T. Ogita, Super-fast Validated Solution of Linear Systems, Journal of Computational and Applied Mathematics (special issue on SCAN2004), to appear.

[12] T. Ogita, S. Oishi, Y. Ushiro: Fast Verification of Solutions for Sparse Monotone Matrix Equations, Topics in Numerical Analysis: With Special Emphasis on Nonlinear Problems (Computing, Supplement 15, G. Alefeld and X. Chen eds.), 175-187, Springer WienNewYork, Austria, 2001.

参考URL

[1] http://planetmath.org/encyclopedia/ToeplitzMatrix.html

[2] http://www.mathworks.com/access/helpdesk/jhelp/techdoc/ref/toeplitz.shtml

[3] http://mathworld.wolfram.com/ToeplitzMatrix.html

[4] http://www.netlib.org/utk/forums/netlib/messages/419.html

[5] http://www.mathworks.com/access/helpdesk/jhelp/toolbox/signal/signal.shtml

[6] http://www.mlab.ice.uec.ac.jp/ mit/text/gsig/2003/Sensing31-39.pdf