精度保証付き数値計算に基づく
検証付き制御系設計に関する研究

矢野　健太郎

目 次

表 目 次

図 目 次

第1章　はじめに

1.1　背景

制御系の設計プロセスは図 1.1 に示すようにモデリング，制御器の設計，シミュレーション，制御実験の流れで行われる．この制御系を設計する際には様々な誤差が混入する．モデリングの際には実際の制御対象と数式モデル間の誤差であるモデル化誤差，その後のプロセスでは計算機を用いる際に打ち切り誤差や数値の丸め誤差等の数値計算誤差が発生する．この誤差の影響により，実際は不安定や不可制御であるはずのシステムが安定や可制御と誤って判定されてしまう可能性がある．また，制御器として，性能が低い，制御器の仕様や性質を満たさない，システムを安定化することができない，ものが求まる可能性がある．このような制御器を用いると，制御に失敗するだけでなく人間や機器に悪影響を与えてしまうかもしれない．

上記の誤差のうちモデル化誤差を考慮した設計法として，ロバスト制御理論 [1, 2] が提案されている．ロバスト制御理論では，実システムとモデルの差の範囲を見積もり，実システムとモデルを含む集合を考える．そして，制御器を設計する際にこの集合について安定性や制御性能を保証することで，実システム及び集合に属する任意のシステムに対する安定性・性能を保証する．ロバスト制御理論の代表的な手法には $\mathcal{H}_\infty$ 制御理論があり，多くのシステムに適用され成果をあげている．

このようにモデル化誤差に対しては，集合を用いることで制御系の安定性・性能を数学的に保証する手法が提案されている．しかし，プロセスのその後の段階で発生する数値計算誤差に関しては計算誤差による問題の指摘 [3] は存在したが，問題を解決する手法は提案されていない．

数値計算誤差には打ち切り誤差や，数値の丸め誤差があり，通常の計算機で数値計算を行う際に発生する．打ち切り誤差は無限次元問題を有限次元化する際や，反復計算を途中で打ち切ることによって発生する．また，数値の丸め誤差は実数を浮動小数点数に近似する際に発生する．この数値計算誤差を見積もる時にも誤差が発生するため，どのくらいの誤差が発生しているかを計算するのは容易ではない．数値計算誤差が原因で発生した事故例を，表 1.1 に示す．

数値計算誤差の問題を解決する手法として，多倍長演算が提案されている．多

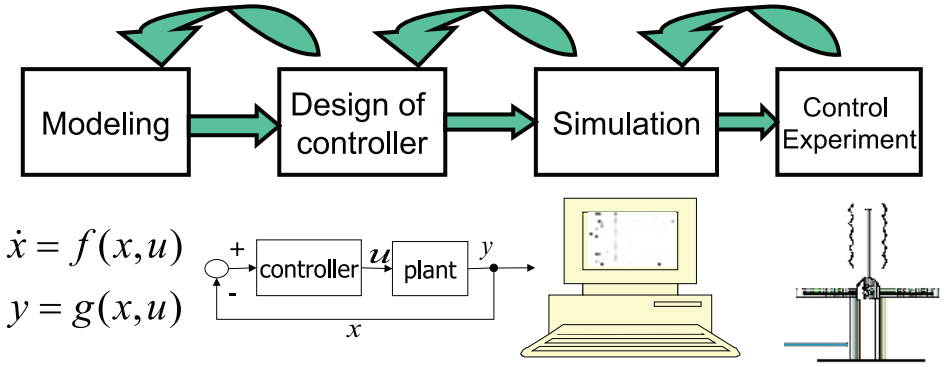

図 1.1: Design process of control system

表 1.1: Accident examples caused by numerical error

事故	原因	詳細
パトリオットミサイルの欠陥	オーバーフロー（約 0.000000095）	24bit 固定点レジスタを使用 1/10 の 2 進表現の際の切捨て 100 時間で 0.34 秒のズレ
Ariane5 の爆発	オーバーフロー	64bit 浮動小数点数→ 16bit 符号付整数
Sleipner A 海上施設の沈没	不正確な有限要素近似	NASTRAN を使用 せん断応力の 47%の過小評価

倍長演算は演算精度（precision）を向上させることで，数値計算誤差の問題を解決する．図 1.2 に浮動小数点数のフォーマットを示す．通常の数値計算に使われる倍精度浮動小数点数は 1 ビットの符合部，11 ビットの指数部，52 ビットの仮数部を持っており，10 進数で約 16 桁の精度を持っている．多倍長精度浮動小数点数は，1 ビットの符合部を持ち，指数部と仮数部を任意に指定できるので，10 進数で任意の精度を持たせることができる．このように多倍長演算は，浮動小数点数仮数部の大きさを任意に指定することで丸め誤差を減少させ，演算精度を向上させる手法である．

このように多倍長演算を用いることで，数値計算を行う際の個々の演算の精度を向上させることができる．しかし，多倍長演算を用いても，近似計算を行ったのでは最終的に得られる結果の精度である結果精度（accuracy）に関する保証はない．

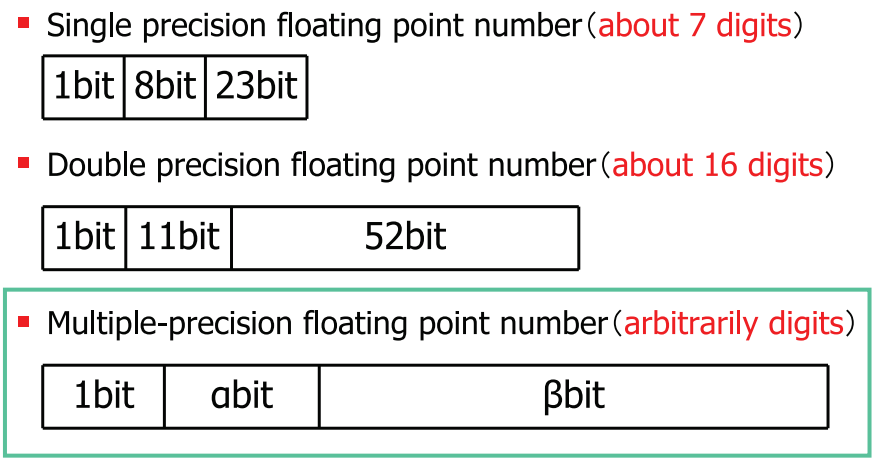

図 1.2: Format of floating point number

結果精度（accuracy）を保証する手法として，計算結果がどのくらい正しいかを検算する精度保証付き数値計算 [4–7] という計算手法が提案されている．精度保証付き数値計算は，真の解と近似解を含む集合を求めることで

- 真の解は存在するのか？
- 近似解の精度はどれくらいなのか？

という問いに答える．まず一つ目の問いに対しては真の解と近似解を含む集合を求めることによって答え，二つ目の問いには求めた集合の大きさによって答える．このように，精度保証付き数値計算を用いることによって計算誤差の大きさを見積もることができる．

この多倍長演算と精度保証付き数値計算の関係を表 1.2 に示す．多倍長演算を用いることで高精度の解を求めることができるが，近似解を求めたのでは計算結果に関する保証はない．一方，精度保証計算を行うことで結果精度保証付きの解を求めることができるが，倍精度演算を用いたのでは低精度な解が求まる．そこで，精度保証付き数値計算と多倍長演算を組み合わせることで高精度かつ結果保証付きの，高品質な数値計算を行うことができる．

ここで，精度保証付き数値計算の歴史について述べる．1950 年代に区間演算が須永や Moore によって提案され，1960 年代には区間解析ブームが起こった．それらの研究では，（ガウスの消去法などの）有限回の四則演算のみで行える計算アルゴリズムにおいて実数を区間，四則演算を区間演算に置き換えることで結果精度を保証した．これは，単純な精度保証付き数値計算の原理である．しかし，1970 年代から 80 年代には低迷期が訪れる．これは単純に区間演算を繰り返すと区間幅

表 1.2: Verified numerical computation and multiple-precision arithmetic

	近似解	精度保証
倍精度	保証なし 低精度	保証付き 低精度
多倍長精度	保証なし 高精度	保証付き 高精度

が増大すること，及び解ける問題の大きさに制限が有ったためである．これにより，精度保証付き数値計算は実用的でないとの評判が残った．しかし，この時期にも地道な改良は続けられ，1990 年代には区間幅の増大を抑制し高速演算を行う手法として，平均値形式，自動微分や IEEE754 で定義された CPU 丸めモード変更を用いる手法が提案された．このように演算精度・速度の改善や適応可能な問題の増加により，精度保証付き数値計算は実用段階に入っている．

精度保証付きで解ける数学の問題の一覧を以下に示す．

- 線形方程式
- 非線形方程式
- 固有値問題
- 関数の微分
- 常微分方程式
  - 初期値問題
  - 境界値問題
- 偏微分方程式
  - 境界値問題

このように，個々の数学の問題に関しては精度保証付きで解く手法が提案されている．そこで，次に制御系設計問題を精度保証付きで解くにはどうすればよいかについて考える．

制御系設計問題の具体例として，LQ 制御問題を考える．LQ 制御問題を通常の手法で解く手順を以下に示す．

1. リカッチ方程式 $A^TP + PA + Q - PBR^{-1}B^TP = 0$ の正定対称な解 P を求める

2. 状態フィードバックゲイン F を $F = R^{-1}B^TP$ から求める

このように制御系設計問題は多段階アルゴリズムであり，ある問題を解いた後にその結果を用いてさらに別の問題を解く，というように複数の数学の問題を組み合わせて解く必要がある．

近年，この精度保証付き数値計算を制御系設計に適用する方法が提案されている [8–18]．たとえば，$\mathcal{H}_2$ 追従制御問題 [10] や，$\mathcal{H}_\infty$ ループ整形問題 [11] を有理数に基づく精度保証付き数値計算で解く方法が提案されている．

このように個々の設計問題に対して，それぞれ個別に精度保証付き数値計算に基づく手法を考える必要がある．その理由は既存のアルゴリズム中の近似計算を単純に精度保証付き数値計算に置き換えただけでは，

- 真の解を含む集合が大きくなる
- 計算に多くの時間がかかる

ためである．したがって，個々の問題について精度保証付き数値計算の視点から問題を見直し，効率的に精度良く解く手法を考案する必要がある．

管野は有理数に基づく精度保証付き数値計算を用いた制御系設計手法 [9–11] を提案している．有理数に基づく手法は計算機を用いて厳密な解を求めることをモチベーションとしており，数式処理に有理数演算と区間演算を組みせることで精度保証を行っている．数式処理を用いるため，問題が解析的に解け，パラメータを残すことができるという特徴がある．しかし，数式処理に基づいているため関係式を導く必要があり，適用できる問題に制限がある．例えば，$\mathcal{H}_2$ 追従制御問題 [10] は一入力一出力の安定な制御対象，$\mathcal{H}_\infty$ ループ整形問題 [11] は一入力一出力の制御対象にしか適応できない．また，数式処理と有理数を用いるために計算速度とメモリ量に関する問題もある．

精度保証付き数値計算には，浮動小数点数に基づく方法もある．この方法は通常の近似計算と同じ浮動小数点数を用いるため，高速に計算を行えるという利点がある．また，通常の近似計算アルゴリズムと精度保証付き数値計算を組み合わせることで，多くの問題を精度保証付きで解くことができる．この方法では精度と計算速度に問題がある可能性があるが，多倍長演算 [19, 20] と精度保証付き数値計算を組み合わせる [21] ことで解決することができる．

上記の手法を用いて精度保証付きで求まった数値解が保証しているのは数値計算の結果精度 (accuracy) であり，設計仕様を満たすことの保証ではない．そこで，設計仕様を満たすことを保証する手法について考える．

通常の近似計算では，制御器の検証は設計した制御器に対して数値計算を行うことで，検証を行う．この検証を行う際に近似計算ではなく，精度保証付き数値計算を用いれば，制御器が設計仕様を満たすことを厳密に保証できる．そこで，本研究では制御器の設計を精度保証付きで行った後に制御器の検証を精度保証付きで行う手法である，検証付き制御系設計手法を提案する．

本研究の目的は，精度保証付き数値計算に基づく検証付き制御系設計手法を提案することである．精度保証付き数値計算を用いて制御器を求め，得られた制御器の検証にも精度保証付き数値計算を用いる．本論文では特に状態フィードバック制御系を検証付きで設計する手法を提案する．具体的な設計問題として，極配置問題 [22, 23] と LQ 制御問題 [22, 23] を取り上げる．LQ 制御問題は実システムへの適応例も多く、実用性に優れている．これまで極配置問題と LQ 制御問題については計算アルゴリズムの精度の比較 [24] や，計算誤差による問題の指摘 [3] は存在したが，精度保証付き数値計算を適用する方法は提案されていない．LQ 制御問題を解くためにはリカッチ方程式を解く必要があり，ハミルトン行列の固有値問題を解く手法 [22] がよく用いられる．しかし，有理数を用いる精度保証方法 [9–11] は一般の行列の固有値と固有空間の基底を求めることができないため，有理数に基づく方法を直接 LQ 制御問題に適応することはできない．

本論文では，以下の手法を提案する．

1. 精度保証付き制御系解析手法
2. 検証付き制御系設計
3. 数値的最適制御問題
4. 数値的最適制御問題の効率的解法
5. 精度保証計算パッケージの実装

制御器の設計を行う前には，システムの安定性解析等の制御系の解析が行われる．そこで，制御系の解析として安定性，可制御性，可観測性の解析を精度保証付き数値計算と多倍長演算 [19, 20] を用いて行う手法 [25] を提案する．精度保証付き数値計算と多倍長演算を用いた高品質な数値計算を用いることにより，制御系の解析を厳密に行うことが可能になる．また，制御系の安定性の解析は行列の正定性に帰着される．行列の正定性の解析が厳密に行えることで，繰り返し計算の終了判定や，多倍長計算をいつ用いるかの判断を厳密に行える．また，制御器の検証にも利用できる．

検証付き制御系設計では、精度保証付き数値計算を用いて求めた制御器が設計仕様を満たしているかの確認を精度保証付きで行う．また，検証付きの制御器の

中から，設計仕様を満たすこと（設計仕様に近い性能）を最も保証できる制御器を選択する方法である，数値的最適制御問題を提案する．本論文では検証付き制御系設計手法と数値的最適制御問題の具体例として，極配置問題と LQ 制御問題の場合について定義する．また，その解法及び解法で必要となる極配置問題及び LQ 制御問題を精度保証付きで解くアルゴリズムを提案する．

数値的最適制御問題を解く際，精度保証付きで求まった解の精度が悪いと大きな半径を持つ区間行列として制御器の集合が求まる．半径の大きな区間行列が求まると数値的最適制御器の候補が多くなってしまい，数値的最適制御器の選択に多くの時間が必要となる．この問題を解決するために，GA[26] を用いて高速に候補集合を作成する方法 [15] と，多倍長演算で区間行列を高精度で求めた後に倍精度演算に近似する手法 [27, 28] を提案する．

また，精度保証付き数値計算を容易に行うために，Java 精度保証付き数値計算パッケージを開発した．開発したパッケージは多倍長演算パッケージと組み合わせることが可能であり，多くの問題を高精度かつ精度保証付きで解くことができる．

表 1.3 に本論文の提案手法と既存の手法の比較を示す．有理数に基づく手法は数式処理を用いており，パラメータを残せるという利点があるが，計算速度が遅い，適応範囲が狭いという欠点がある．本論文で提案する手法は，通常の近似計算アルゴリズムに精度保証付き数値計算を組み合わせるものであり，高速に計算できる，適応範囲が広いという利点がある．また，精度保証付き数値計算を用いて仕様の検証を行い，検証付きの制御器の中から最適解の選択を行うという特徴がある．

表 1.3: Comparison of computation method

	有理数（管野）	浮動小数点数（提案手法）
計算方法	数式処理＋有理数＆区間	近似計算アルゴリズム＋精度保証
パラメータ	残せる	残せない
計算速度	低速	高速
適応範囲	狭い	広い
仕様検証	妥当な解	有り
最適解の選択	なし	有り

1.2 論文の構成

以下に本論文の構成を示す．まず第 2 章で精度保証付き数値計算について述べる．第 3 章で精度保証付き制御系解析手法，第 4 章で検証付き制御系設計手法に

ついて述べる．第5章では，検証付き制御器の集合の中から最も設計仕様を満たすものを選択する手法である数値的最適制御問題を効率的に解く手法を提案する．第6章に本論文で提案した手法をJavaとC言語を用いて実装した例を示し，第7章でまとめる．

1.3 論文中の記法

本論文では以下の記法を用いる．I は単位行列，$\boldsymbol{R}^n$ は n 次元実ベクトルの全体，$\boldsymbol{R}^{n\times n}$ は n 次の実正方行列の全体，$I\boldsymbol{R}^n$ は n 次元実区間ベクトルの集合，$I\boldsymbol{C}^n$ は n 次元複素区間ベクトルの集合，$[x]=[\underline{x},\overline{x}]=\{x|\underline{x}\leqq x\leqq\overline{x}\}$，$||[x]||=\max\{|x||x\in[x]\}=\max\{|\underline{x}|,|\overline{x}|\}$ である．

第2章　精度保証付き数値計算

2.1　精度保証付き数値計算の原理

ここでは問題の解を精度保証付きで求めるための原理と手法をごく一般的な形において説明する [6].

精度保証付き数値計算は以下に示す手順で行われる.

(S1) 真の解を含む集合の候補（候補集合）を何らかの方法で決定

(S2) 候補集合が真の解を含むことを（不動点定理を用いて）検証

(S3) 真の解を含まなければ，別の候補（集合）を立てて，前段に戻る

(S4) 真の解を含んでいれば，解の精度がその集合の大きさによって決定され，計算終了

候補集合が真の解を含むことを確認する際に使われる不動点定理の一般的な表現を以下に示す [6].

ある条件を満たす集合 U に対し $F(U)$ を

$$F(U) := \{v \mid v = F(u), u \in U\}$$

とする．このとき集合 U が

$$F(U) \subset U$$

を満たせば U の中に $\hat{x} = F(\hat{x})$ を満たす $\hat{x}$ が存在する.

精度保証付き数値計算で用いられる不動点定理の例を以下に示す.

- 線形方程式
  - Brouwer の不動点定理
- 常微分方程式

  - Banach の不動点定理
  - Schauder の不動点定理

- 偏微分方程式
  - Schauder の不動点定理

ここでは不動点定理を用いた精度保証付き計算の具体例として，n 次の線形方程式 $Ax = b$, $A \in \boldsymbol{R}^{n\times n}$, $b \in \boldsymbol{R}^n$ の解の精度保証を考える．線形方程式の精度保証には，Brouwer の不動点定理から導かれるつぎの系が使われる．

系 1 *[4, 6]* ある $R \in \boldsymbol{R}^{n\times n}$ と $X \in I\boldsymbol{R}^n$ に対して

$$Rb + \{I - RA\} \cdot X \subset int(X)$$

が成り立てば，R, A はともに正則行列で，$A\hat{x} = b$ なるただ *1* つの元 $\hat{x} \in int(X)$ が存在する．ただし，$int(X)$ は集合 X の内部（内点全体）を表す．

系 1 を用いることで候補集合 X が真の解 $\hat{x}$ を含んでいるかを確認することができる．また，系 1 で通常 R は A^{-1} の近似解に取られる．

上記の系では集合として区間を用いている．これにより，真の解の存在する区間を求めることができる．そして，求めた値をその後の計算で用いる場合には，求めた区間に対して演算を行うことで精度を保証する．このように精度保証付き数値計算は，浮動小数点数の表す数値の代わりに区間に対して計算を行う区間演算 [4–7, 29] を行うことで，問題の真の解を区間の中に包み込むという考え方で行われる．

2.2　区間演算

本節では，前節で述べた区間演算についてより詳しく述べる．

２つの区間 $X = [x] = [\underline{x}, \overline{x}], Y = [y] = [\underline{y}, \overline{y}]$ が与えられたとき，その二つの区間の四則演算を次のように定義する．

$$X \circ Y = \{x \circ y | x \in X, y \in Y\}$$

ただし，$\circ \in \{+, -, \times, /\}$ である．また，除法 / の場合は Y が 0 を含まないものとする．これを区間演算という．この定義では，以下が成立する．

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

区間演算においては，包含関係における単調性

$$\begin{aligned}
[x] \subseteq [x'], [y] \subseteq [y'] \Rightarrow [x] \circ [y] \subseteq [x'] \circ [y'] \\
\circ \in \{+, -, \times, \div\}
\end{aligned}$$

が成立する．また，加法と乗法に関して以下が成立する．

$$\begin{cases}
[x] \circ [y] = [y] \circ [x], \circ \in \{+, \times\} \\
([x] \circ [y]) \circ [z] = [x] \circ ([y] \circ [z]), \circ \in \{+, \times\}
\end{cases}$$

区間演算に関して分配則は成立はせず，次の劣分配則が成立する．

$$[x] \times ([y] + [z]) \subseteq ([x] \times [y]) + ([x] \times [z])$$

2.3 区間演算のみを用いる精度保証付き数値計算

四則演算のみで計算できる規模の小さな問題であれば，区間演算のみを用いて精度保証を行うことができる．この節では，区間演算のみを用いて精度保証を行う問題について述べる．

2.3.1 微分値の精度保証

解きたい問題が線形でない場合，非線形方程式を解かなければならない．非線形方程式の解法としては，ニュートン法などが有名であるが，ニュートン法を実行するためには，ヤコビ行列を計算する必要がでてくる．そこで，関数を計算するプログラムから関数の微分を計算するプログラムを自動生成できれば望ましい．これを自動微分 [4, 5, 30, 31] という．ここでは，基本的な微分可能関数の四則演算や合成関

数として得られる関数のクラスに対して，その微分を計算する方法を示す．今，独立変数 t の計算可能単項関数 $x_i(t), (i = 1, 2, \cdots, n)$ の導関数 $\dot{x}_i(t), (i = 1, 2, \cdots, n)$ が再び計算可能であるとする．計算可能関数 $x(t)$ が有限長プログラム P_x によって定義されているとする．このプログラムは，計算可能単項関数 $x_i(t), (i = 1, 2, \cdots, n)$ を用いて，その四則演算と合成関数をとる操作を繰り返し用いることによって定義されているものとする．自動微分型オブジェクトは，関数値 $f(t)$ とその微分値 $\dot{f}(t)$ の組 $(f(t), \dot{f}(t))$ として定義される．二つの自動微分型に対してその四則演算は，

$$\begin{aligned}
(f, \dot{f}) + (g, \dot{g}) &= \left(f + g, \dot{f} + \dot{g}\right) \\
(f, \dot{f}) - (g, \dot{g}) &= \left(f - g, \dot{f} - \dot{g}\right) \\
(f, \dot{f}) \times (g, \dot{g}) &= \left(f \times g, \dot{f} \times g + f \times \dot{g}\right) \\
(f, \dot{f}) / (g, \dot{g}) &= \left(\frac{f}{g}, \frac{\dot{f} \times g + f \times \dot{g}}{g^2}\right)
\end{aligned}$$

と定義される．また，自動微分型 $(f(t), \dot{f}(t))$ に単項演算 x_i を作用させた結果として，得られる自動微分型は $(g(t), \dot{g}(t)) = (x_i(f(t)), \dot{x}_i(f(t))\dot{f}(t))$ と定義される．プログラム P_x に $(t, 1)$ を代入すればその出力は $(x(t), \dot{x}(t)))$ となる．このようにして，関数の $x(t)$ の値 t における正確な微分値 $\dot{x}(t)$ が自動的に得られる．

上記の自動微分と区間演算を組み合わせることにより，数値の丸め誤差までを含め微分値を厳密に評価することができる．

2.4　不動点定理を用いる精度保証付き数値計算

区間演算のみを用いてある程度規模の大きな問題の精度保証を行おうとすると，計算速度や区間の幅（精度）に関する問題が発生する．そこで，本節では不動点定理を用いた解の存在の保証と計算の効率化に基づく精度保証手法について述べる．

2.4.1　非線形方程式の精度保証

ここでは非線形方程式の精度保証に用いられる，Newton 法の区間演算版であるクラフチック法について述べる．$f : \boldsymbol{R}^n \to \boldsymbol{R}^n$ を C^1 級とし，非線形方程式を $f(x) = 0$ と書く．また，f の x におけるヤコビ行列を $f'(x)$ とし，$X \in I\boldsymbol{R}^n$ に対し，$F'(X)$ を区間演算で $f'(x)$ を評価したものとする．このとき，Krawczyk-Moore の作用素として次式を定義する．

$$K(X) := \tilde{x} - Yf(\tilde{x}) + \{I - YF'(X)\}(X - \tilde{x})$$

ここで，$\tilde{x}$ は与えられた X 内の点で，Y は任意の正則行列，たとえば $(f'(x))^{-1}$ の近似である．また，区間 X を与える代わりに通常の Newton 法で求めた近似解 $\tilde{x}$ を与え，$\tilde{x}$ を含む区間 X を適当な方法で計算しても良い．もし

$$K(X) \subset X \text{ および } ||I - YF'(X)|| < 1$$

が成り立つならば，非線形方程式 $f(x) = 0$ の解が X 内にあり，反復列:

$$X^{(k+1)} := X^{(k)} \cap K(X^{(k)})(k = 0, 1, \cdots), \quad X^{(0)} = X$$

は，$X^{(0)}$ 内にある方程式の解に収束することが示される．ただし，$||\cdot||$ は行列ノルムを表す.

この Krawczyk-Moore 作用素を繰り返し適用することにより，$X^{(k)}$ に非線形方程式の解を含んだまま，区間 $X^{(k)}$ の半径を小さくすることができる.

2.4.2 固有値問題の精度保証

ここでは，一般的な行列に対する固有値問題 $Av = \lambda v$ の精度保証方法 [32, 33] について述べる.

はじめに，この定理の説明で使用する記号を説明する．$\boldsymbol{K}$ は実数 $\boldsymbol{R}$ もしくは，複素数 $\boldsymbol{C}$ とする．また，$A \in M_n(\boldsymbol{K})$ を $n \times n$ の行列とし，$\tilde{X} \in M_{n,k}(\boldsymbol{K})$ を $A\tilde{X} \approx \tilde{\lambda}\tilde{X}$ であるような不変部分空間の基底の近似とする．以下，n を行列の次元、k を不変部分空間の次元とする．さらに，行列 U と V を以下のように定義する.

$U \in M_{n,n-k}(\boldsymbol{K})$ は単位行列からある k 個の列を取り去ったものとし，$V \in M_{n,k}(\boldsymbol{K})$ は U で取り去ったある k 個以外の単位行列 $n - k$ 列で構成される行列である．これらは $UU^T + VV^T = I_n$ を満たす.

単一の正規化条件付きの固有値問題

$$\begin{cases} Ax = \lambda x \\ x_i = \xi \end{cases} \tag{2.1}$$

を次のように拡大する.

$$\begin{cases} AY = YM \\ V^TY = V^T\tilde{X} \end{cases}$$

そして，この問題に対して固有値に対する不変部分空間の包み込みを目指すことを考える．ここで $V^TY = V^T\tilde{X}$ は，式 (2.1) の固有値問題における拘束条件 $x_i = \xi$ にあたり，$k \times k$ 個の要素を拘束している.

この問題に必要な補題を説明する.

補題 1 $\mathbf{Z}, \mathbf{X} \in I\boldsymbol{K}^n$, $\boldsymbol{C} \in IM_n(\boldsymbol{K})$ が与えられたとする．このとき，

$$\mathbf{Z} + \mathbf{C} \cdot \mathbf{X} \subseteq \mathrm{int}(\mathbf{X})$$

であれば，各 $C \in \mathbf{C}$ は収束する．ただし，$\mathrm{int}(X)$ は集合 X の内部（内点全体）を表す．

この補題から次の定理が導かれる．

定理 1 $A \in M_n(\boldsymbol{K}), \tilde{X} \in M_{n,k}(\boldsymbol{K}), R \in M_n(\boldsymbol{K})$, そして $\mathbf{X} \in IM_{n,k}(\boldsymbol{K})$ とする．また，U, V を $UU^T + VV^T = I_n$ となるような行列とする．いま，

$$f(\mathbf{X}) := -R(A\tilde{X} - \tilde{\lambda}\tilde{X}) + \{I - R((A - \tilde{\lambda}I)UU^T - (\tilde{X} + UU^T\mathbf{X})V^T)\}\mathbf{X}$$

とし，

$$f(\mathbf{X}) \subseteq \mathrm{int}(\mathbf{X}) \tag{2.2}$$

であるならば，このとき

$$A\hat{Y} = \hat{Y}\hat{M}$$

であるような $\hat{M} \in \tilde{\lambda}I_k + V^T\mathbf{X} \in M_k(\boldsymbol{K}), \hat{Y} \in \hat{X} + UU^T\mathbf{X} \in M_{n,k}(\boldsymbol{K})$ が存在する．

上の定理において，$\tilde{\lambda}I_k + V^TX$ の k 個の固有値は，A のある k 個の固有値と一致する．また，$\tilde{X} + UU^T\mathbf{X}$ の各列ベクトルはそれら k 個の固有値に対する不変部分空間の基底をなす．この問題では，解の行列 $\mathbf{X}$ の中に Y, M の要素を同時に包み込むようなアルゴリズムを考えている．

次に，$\tilde{X}, \tilde{\lambda}, R$ の求め方について考える．これらには，精度保証を行うための前提となる仮定が存在しないことが，この方法の優れた点である．唯一の仮定が，式 (2.2) である．特に近似解に関する前提仮定はないので，必要に応じた方法で近似解を求めればよい．

次に精度保証を実現するために，区間行列 $\mathbf{X}$ を求めるための区間反復について説明する．基本的には，区間反復は $\mathbf{X}^{v+1} = f(\mathbf{X}^v)$ で与えられる．しかし，$\mathbf{X}$ が式 (2.2) を満たすようにするためには，$\mathbf{X}$ にある程度の幅が必要である．これを達成する適切な方法が epsilon-inflation である．これは具体的には，次のアルゴリズムで与えられる．

$R = ((A - \tilde{\lambda}I)UU^T - \tilde{X}V^T)^{-1}$;
$\mathbf{Z} = -R(A\tilde{\mathbf{X}} - \tilde{\lambda}\tilde{\mathbf{X}})$;

$\mathbf{C} = I - R((A - \tilde{\lambda}\mathbf{I})UU^T - \tilde{X}V^T)$;
$\mathbf{X} = \mathbf{Z}; \alpha = 0; \alpha_{\max} = 10$;
repeat
  $\alpha = \alpha + 1$;
  $\mathbf{Y} = \mathbf{X} + 0.1 \cdot [-|\mathbf{Z}| - \epsilon, |\mathbf{Z}| + \epsilon]$;
  $\mathbf{X} = \mathbf{Z} + \mathbf{CY} + R(UU^T\mathbf{Y}V^T\mathbf{Y})$;
  $ready = (\mathbf{X} \in \mathrm{int}(\mathbf{Y}))$;
until $ready$ *or* $(\alpha = \alpha_{\max})$;
if ready
  for $i = 1 : k$
    $r = \sum_{j\neq} |(V^T\mathbf{X})_{ij}|$;
    $\mathbf{L}_i = \mathrm{complex}(\tilde{\lambda} + (V^T\mathbf{X}_i)i + [-r, r])$;

上のアルゴリズムにおいて，$\tilde{\mathbf{X}}$ は $\tilde{X}$ を包み込むような小さな区間行列，$\mathbf{I}$ は単位行列 I を包み込むような小さな区間行列である．また，反復の初期値 $\mathbf{X}_0$ としては，$-R(A\tilde{X} - \tilde{\lambda})$ を包み込むような小さな区間をとる．ϵ は浮動小数点数の正の最小の値である．また，$\mathbf{L}_i$ は行列 $\mathbf{M} := \tilde{\lambda}I_k + V^T\mathbf{X}$ に対するゲルシュゴリン円板を計算している．

2.4.3 多項式の評価の精度保証

多項式 $p : \boldsymbol{R} \to \boldsymbol{R}$

$$p(t) = \sum_{i=0}^{n} p_i t^i$$

を考える [5, 30]．ここで，$p_i, t \in \boldsymbol{R}$ である．また，$p_n = 0$ とする。多項式 p の値を点 $t \in \boldsymbol{R}$ で計算する問題を考える．通常，多項式の値の評価にはホーナー法が用いられる．これは，$p(t)$ を

$$p(t) = ((\cdots((p_n t + p_{n-1})t + p_{n-2})t + \cdots p_2)t + p_1)t + p_0$$

の形に変形して計算する方法である．すなわち，ホーナー法は

$$\begin{cases} x_n = p_n \\ x_i = x_{i+1}t + p_i \quad (i = n-1, \cdots, 0) \end{cases}$$

と計算したとき，$p(t) = x_0$ と計算する方法である。

実数上でホーナー法を用いれば真の値が計算できるが，浮動小数点数システム上では近似値の計算となる．その計算誤差を評価することを考える．$n+1$ 次元ベクトル $x = (x_n, x_{n-1}, \cdots, x_0)^t$, $p = (p_n, p_{n-1}, \cdots, p_0)^t$ を導入する．また，$(n+1)\times(n+1)$ 行列 A を

$$A = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ -t & 1 & & \vdots & 0 \\ 0 & 0 & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & -t & 1 \end{bmatrix}$$

とする．このとき，ホーナ法の計算過程は次の連立一次方程式方程式の形に変形することができる．

$$Ax = p$$

x_0 が多項式 $p(t)$ の値である．

浮動小数点数システム $\mathbf{F}$ 上で，ホーナー法により次のように計算した x を $\tilde{x} = (\tilde{x}_n, \cdots, \tilde{x}_1)^t$ とする．

$$\begin{cases} \tilde{x}_n = p_n \\ \tilde{x}_i = \tilde{x}_{i+1}t' + p_i \quad (i = n-1, \cdots, 0) \end{cases}$$

ただし，$t' \in [t_c - d, t_c + d]$ とする．$d \geqq 0$ は t の丸め誤差であり，係数も丸め誤差などの影響で $p_n \in [\underline{p}_n, \overline{p}_n]$ と区間評価されているとする．残差 $[r]$ を

$$[r] \subseteqq [p] - [A]\tilde{x}$$

と計算する．ただし，$[A]$ は $[t] = [t_c - d, t_c + d]$ で置き換えられた区間行列である．

行列 A の要素 t を t' で置き換えた行列を A' とすると，A' の逆行列は

$$A'^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ t' & 1 & & \vdots & 0 \\ t'^2 & t' & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 \\ t'^n & t'^{n-1} & \cdots & t' & 1 \end{bmatrix}$$

で与えられる．よって，その作用素ノルムは

$$\|A'^{-1}\|_\infty = \sum_{i=0}^{n} |t'|^i = \frac{1 - |t'|^{n+1}}{1 - |t'|}$$

で与えられる．また，

$$g = \|I - A'^{-1}A\|_\infty \leqq d \max\{1, |t'|^{n-1}\}$$

であるから山本の定理 [5] より

$$p(t) \in \tilde{x}_0 + \frac{(1 - |t'|^{n+1})/(1 - |t'|)}{1 - g}[\underline{r}_0, \overline{r}_0]$$

と評価される．

第3章　精度保証付き制御系解析

数値計算誤差の影響により，実際は不安定や不可制御であるはずのシステムが安定や可制御と判定されてしまう可能性がある．本章では制御系解析に関する計算に浮動小数点数に基づく精度保証付き数値計算を応用し，制御系解析を精度保証付きで行う手法を提案する [25]．本章では，制御系解析の計算，特に安定性の解析，可制御性，可観測性における計算結果の品質保証を行う手法を提案する．また，制御系の安定性の解析は行列の正定性に帰着される．行列の正定性の解析が厳密に行えることで，繰り返し計算の終了判定や，多倍長計算をいつ用いるかの判断を厳密に行える．また，制御器の検証にも利用できる．

3.1　行列の正定性

まずはじめに，制御系解析の基礎となる行列の正定性の判定について述べる．通常の数値計算では行列の固有値を求め，全ての固有値の実部が負，つまり複素数平面上で右半平面にあれば正定，左半平面にあれば不定と判定する．

精度保証付き数値計算を用いた行列の正定性の解析の概要を，図 3.1 に示す．精度保証付き数値計算で固有値を求めた場合，図 3.1 に示すように固有値の真の値を含む集合が求まる．そして，この集合が左半平面のみにある場合は固有値が左半平面に存在するため，その行列は負定である．同様に，集合が右半平面のみに存在する場合は正定となる．しかし，集合が左半平面と右半平面にまたがって存在する場合には真の値が左半平面にあるか，右半平面にあるかを特定できない．そこで，このときその行列は正定とはいえない（正定非保証）と呼ぶことにする．なお，従来の計算法ではこのような行列は正定であるにもかかわらず計算途中の誤差の影響によって負定であると判定されていた可能性がある．つまり，精度保証付き数値計算を適用することにより正定性解析における数値的品質を保証することが可能である．

また，正定とはいえない結果が求まった際には多倍長演算を行うことで演算精度を上げ，集合の大きさを小さくすることができる．そうすることで集合が左右どちらかの領域のみに存在することが分かれば，より厳密な判定を行うことも可能である．

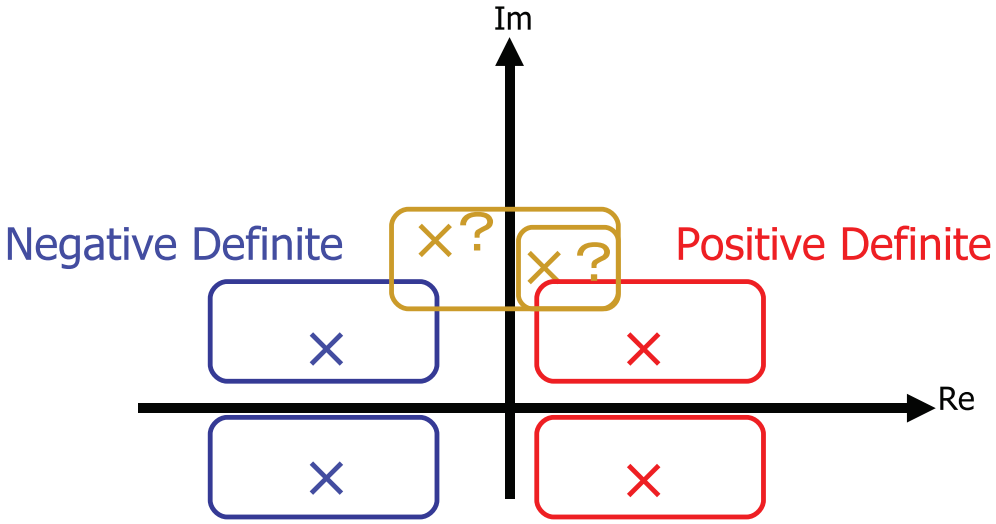

図 3.1: Analysis of positive definite

3.2　安定性解析

安定性解析は通常の数値計算ではシステムの極 (システム行列の固有値) を求め，その値の実部が負，つまり複素数平面上で左半平面にあれば安定，右半平面にあれば不安定と判定する [22]．そこで，精度保証付き数値計算に基づく安定性の解析は行列の正定性の判定と同様に，システムの極の真の値を含む集合を求め，その集合が左半平面のみにあればそのシステムは安定，集合が右半平面のみに存在する場合は不安定となる．そして，左半平面と右半平面にまたがる集合が存在する場合には，そのシステムは安定とはいえない（安定非保証）と呼ぶことにする．従来の計算法ではこのようなシステムは不安定であるにもかかわらず計算途中の誤差の影響によって安定であると判定されていた可能性がある．つまり，精度保証付き数値計算を適用することにより安定性解析における数値的品質を保証することが可能となる．

また，安定とはいえない極が求まった際には多倍長演算を行うことで集合を小さし，より厳密な判定を行うことが可能である．

3.3　可制御性解析

システムが可制御であるための必要十分条件は，次の可制御行列

$$U_c = \left[B, AB, A^2B, \cdots, A^{n-1}B\right]$$

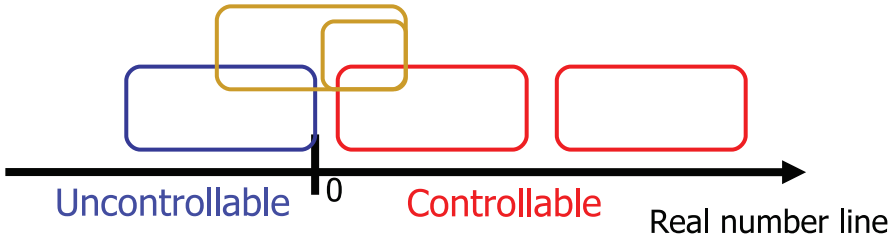

図 3.2: Analysis of controllability

の階数 (rank) がシステムの次数と等しいことである [22].

この rank の計算には，特異値が用いられる．可制御性行列 U_c の特異値を求め，その個数を数えることによって rank を求めることができる．また，行列 U_c の特異値は $U_cU_c^T$ の非零固有値の平方根の個数から求めることができる．

そこで精度保証付き数値計算を可制御性の解析に適用すると，可制御性行列の特異値を $U_cU_c^T$ の非零固有値の平方根から求め，その結果の n 個の集合が複素数平面の右半平面のみに存在すれば，そのシステムは可制御となる．また，左半平面のみに存在する集合が存在する場合は不可制御となる．左半平面と右半平面にまたがる集合が存在する場合には，可制御ではないとはいえない（可制御非保証）と呼ぶ．この場合，安定性解析と同様に，多倍長演算を行うことで集合を小さくし厳密な判定を行うことができる．

精度保証付き数値計算に基づく可制御性解析を，図 3.2 に示す．

3.4 可観測性解析

システムが可観測であるための必要十分条件は，次の可観測行列

$$U_o = \left[C^T, A^TC^T, \left(A^T\right)^2 C^T, \cdots, \left(A^T\right)^{n-1} C^T\right]$$

の階数 (rank) がシステムの次数 n と等しいことである [22]．可制御性の判定と同じことが可観測性の判定にも言える．

3.5　数値例

3.5.1　正定性

次の正定な行列の正定性を判定する．

$$A = \begin{bmatrix} 1 \times 10^{-13} & 0 & 0 \\ 0 & 1 \times 10^{-8} & 0 \\ 0 & 0 & 1 \times 10^{6} \end{bmatrix}$$

この行列の固有値は，$\{1 \times 10^{-13}, 1 \times 10^{-8}, 1 \times 10^{6}\}$ である．この行列の固有値を数値計算で求めると，数値計算は行わずに行列の対角性から対角成分をそのまま固有値とする可能性がある．そこで，PAP^{-1} の固有値を計算することで，この行列の正定性を判別する．ただし，

$$P = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

である．

まず，この行列の固有値を Matlab[34] を用いて近似計算で行った．ただし，Matlab のバージョンは Matlab 7.4.0(R2007a) であり，固有値の計算には eig を用いた．そのときの結果は，

```
ans =

    1.000000000000000e+006
   -1.656778416991734e-010
    9.937794293366050e-009
```

となり，負の値が求まっていることが分かる．この結果より，通常の近似計算による結果ではこの行列は本当は正定であるにもかかわらず，不定であると判定されてしまう．

次に，この問題を文献 [32] のアルゴリズム（2.4.2 節参照）を実装したプログラムを用いて精度保証付きで解いた結果を以下に示す．

```
===  下限  (   3  x   1)  CoMatrix  ===
       [        (  1)-Real                    (  1)-Imag        ]
 (  1)   9.9999999999999700000e+05 -2.5834825877790930000e-10
 (  2)   9.7766383408032570000e-09 -2.7394613205460080000e-10
 (  3)  -2.2352080218910518000e-10 -1.8608639915452434000e-10
 ===  上限  (   3  x   1)  CoMatrix  ===
       [        (  1)-Real                    (  1)-Imag        ]
 (  1)   1.0000000000000001000e+06  2.5834825877790930000e-10
 (  2)   1.0324530604912460000e-08  2.7394613205460080000e-10
 (  3)   1.4865199611994350000e-10  1.8608639915452434000e-10
```

上記の結果は，区間の中に真の解が存在することを保証している．しかし，実部が正と負にまたがる区間が存在している．よって，上記の結果は不定の可能性があること（正定非保証）を示している．

そこで，この問題を多倍長演算パッケージ [20] を用いて多倍長精度保証付きで解いた結果を以下に示す．用いた桁数は 10 進数で 115 桁である．ただし，スペースの都合から始めの 17 桁だけを示す．

```
===  下限  (   3  x   1)  NumericalComplexMatrix  ===
       [                                    (  1)
 (  1)  9.99999999999999999e+5,  -4.24585183943243523 41e-109
 (  2)  9.99999999999999999e-9,  -2.95766004323309737 24e-109
 (  3)  9.99999999999999999e-14, -3.32564884429252792 64e-109

===  上限  (   3  x   1)  NumericalComplexMatrix  ===
       [                                    (  1)
 (  1)  1.00000000000000000e+6,  4.24585183943243523414e-109
 (  2)  1.00000000000000000e-8,  2.95766004323309737248e-109
 (  3)  1.00000000000000000e-13, 3.32564884429252792644e-109
```

上記の結果は実部の区間が正の範囲だけに存在しており，行列が正定であることを示している．このように，精度保証付き数値計算と多倍長演算を用いた高品質数値計算を用いることで行列の正定性を正確に判定できることが分かる．

3.5.2　安定性解析

以下に示す不安定なシステムを考える．

$$A = \begin{bmatrix} 1\times10^{-15} & 5 & 0 & 0 & 0 & 0 \\ -5 & 1\times10^{-15} & 0 & 0 & 0 & 0 \\ 0 & 0 & -1\times10^{-15} & 1 & 0 & 0 \\ 0 & 0 & -1 & -1\times10^{-15} & 0 & 0 \\ 0 & 0 & 0 & 0 & -1\times10^{6} & 1 \\ 0 & 0 & 0 & 0 & -1 & -1\times10^{6} \end{bmatrix}$$

この行列 A の固有値は，$\{1\times10^{-15}\pm5i, -1\times10^{-15}\pm i, -1\times10^{6}\pm i\}$ である．PAP^{-1} の固有値を計算することで，このシステムの安定性を判別する．ただし，

$$P = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 4 & 5 & 6 & 1 \\ 3 & 4 & 5 & 6 & 1 & 2 \\ 4 & 5 & 6 & 1 & 2 & 3 \\ 5 & 6 & 1 & 2 & 3 & 4 \\ 6 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

である．

まず，この行列の固有値を Matlab を用いて近似計算で行った．ただし，Matlab のバージョンは Matlab 7.4.0(R2007a) であり，固有値の計算には eig を用いた．

```
ans =

   -1.000000000000000e+006 +9.999999994598671e-001i
   -1.000000000000000e+006 -9.999999994598671e-001i
   -7.193839968167026e-011 +4.999999999915726e+000i
   -7.193839968167026e-011 -4.999999999915726e+000i
   -1.067682791136559e-010 +9.999999999513519e-001i
   -1.067682791136559e-010 -9.999999999513519e-001i
```

上記の結果より全ての固有値が負なので，このシステムは本当は不安定であるにもかかわらず，安定であると判定されてしまう．

次に，この問題を文献 [32] のアルゴリズム（2.4.2 節参照）を実装したプログラムを用いて精度保証付きで解く．

```
===  下限  (   6  x   1)  CoMatrix  ===
        [ (  1)-Real      (  1)-Imag ]
 (  1) -4.500481476049808000E-09  4.999999995556542000E+00
 (  2) -4.500481476049808000E-09 -5.000000004592676000E+00
 (  3) -3.619388657009296700E-09  9.999999964840166000E-01
 (  4) -3.619388657009296700E-09 -1.000000003815167300E+00
 (  5) -1.000000000000004400E+06  9.999999957079747000E-01
 (  6) -1.000000000000004400E+06 -1.000000004584910700E+00
 ===  上限  (   6  x   1)  CoMatrix  ===
        [ (  1)-Real      (  1)-Imag ]
 (  1)  4.535651901626914500E-09  5.000000004592676000E+00
 (  2)  4.535651901626914500E-09 -4.999999995556542000E+00
 (  3)  3.711761882562505000E-09  1.000000003815167300E+00
 (  4)  3.711761882562505000E-09 -9.999999964840166000E-01
 (  5) -9.999999999999953000E+05  1.000000004584910700E+00
 (  6) -9.999999999999953000E+05 -9.999999957079747000E-01
```

上記の結果は，区間の中に真の解が存在することを保証している．しかし，実部が正と負にまたがる区間が存在している．よって，上記の結果は安定非保証であり，不安定の可能性があることを示している．

そこで，この問題をこの問題を多倍長演算パッケージ [20] を用いて多倍長精度保証付きで解いた結果を以下に示す．用いた桁数は 10 数で 115 桁である．ただし，スペースの都合から始めの 17 桁だけを示す．

```
===  下限  (   6  x   1)  NumericalComplexMatrix  ===
       [                                      (  1)
 (  1)  9.9999999999999999e-16, 4.9999999999999999e+0
 (  2)  9.9999999999999999e-16,-5.0000000000000000e+0
 (  3) -1.0000000000000000e-15, 9.9999999999999999e-1
 (  4) -1.0000000000000000e-15,-1.0000000000000000e+0
 (  5) -1.0000000000000000e+6,  9.9999999999999999e-1
 (  6) -1.0000000000000000e+6, -1.0000000000000000e+0
===  上限  (   6  x   1)  NumericalComplexMatrix  ===
       [                                      (  1)
 (  1)  1.0000000000000000e-15, 5.0000000000000000e+0
 (  2)  1.0000000000000000e-15,-4.9999999999999999e+0
 (  3) -9.9999999999999999e-16, 1.0000000000000000e+0
 (  4) -9.9999999999999999e-16,-9.9999999999999999e-1
 (  5) -9.9999999999999999e+5,  1.0000000000000000e+0
 (  6) -9.9999999999999999e+5,- 9.9999999999999999e-1
```

上記の結果より，実部が正の領域のみに存在する区間があることから，このシステムは不安定であることを示している．このように，精度保証付き数値計算と多倍長演算を用いた高品質数値計算を用いることでシステムの安定性を正確に判定できることが分かる．

第4章　検証付き制御系設計

精度保証付き数値計算を用いて制御系を設計することで，計算誤差の問題を解決した精度保証付き制御系を設計することができる．そして，求まった集合の要素である制御器が制御器としての条件を満たすかどうかを確認する，制御器検証問題を定義する．制御器の検証を行う際にも，精度保証付き数値計算を用いる．この手法を用いることによって，検証付き制御器を求めることができる．本章では，この検証付き制御系設計手法について述べる．

また，制御問題を精度保証付きで解くと制御器の集合が得られるが，実際に制御器を実装する際には1個の制御器を決める必要がある．そこで，設計仕様を満たすこと（設計仕様に近い性能）を最も保証できるものを選択する数値的最適制御問題を定義し，その解法を提案する．

4.1　制御器検証問題

本節では，一般的な場合について制御器検証問題を定義する．制御器の評価を精度保証付き数値計算を用いて行うと，制御器の保証できる性質が集合として求まる．そこで，制御器の満たすべき条件をC，制御器の保証できる性質（集合）をSとする．このとき，制御器検証問題はつぎのように与えられる．

問題 1 (制御器検証問題) CとSが与えられたとき，Cを満たす$s \in S$が存在するか否かを調べよ．

もしこのようなsが存在すれば，その制御器は制御器の満たすべき条件を満たさないとはいえないので，その制御器は妥当でないとはいえないと判定される．そして，もしこのようなsが存在しなければ，その制御器は制御器の満たすべき条件を満たさないために，妥当でないと判定される．また，SがCに含まれるとき，その制御器は制御器の満たすべき条件を満たすので，その制御器は妥当であると判定される．図4.1に制御器検証問題の概要を示す．

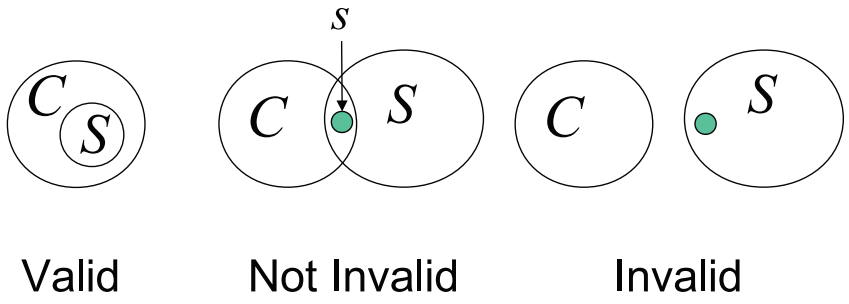

図 4.1: Schematic of controller verification problem

4.2　数値的最適制御問題

数値的最適制御問題は精度保証付きで求まった制御器の集合から，検証付きの制御器を選択し，その中から設計仕様を満たすこと（設計仕様に近い性能）を最も保証できるものを選択する問題である．数値的最適制御問題はつぎのように与えられる．

問題 2 (数値的最適制御問題)

$$\min_{\tilde{F}\in\mathcal{F}} K(\tilde{F}) \tag{4.1}$$

$$subject\ to\ C$$

ただし，$\mathcal{F}$ は与えられた制御系設計問題を精度保証付きで解いて得られた制御器の集合，$\tilde{F}$ は $\mathcal{F}$ 内の値を選択して作成した制御器（数値解），$K(\tilde{F})$ は設計仕様の非適合度に関する式，C は制御器の満たすべき条件である．

数値的最適制御問題の評価関数 $K(\tilde{F})$ は，設計仕様の非適合度に関する式である．$K(\tilde{F})$ は $\tilde{F}$ を用いて設計仕様に関する式を精度保証付きで計算し，その結果（集合）と与えられた設計仕様（集合）の中で最も離れている要素間の距離を求める．そして，制御器の検証条件を満たし，$K(\tilde{F})$ を最小にする $\tilde{F}$ を $\mathcal{F}$ の中から探すことで，検証付きかつ設計仕様を満たすことを最も保証できる制御器（数値解）を選択することができる．

数値的最適制御問題を解くアルゴリズムを以下に示す．

(S1) 精度保証付き数値計算で求めた制御器の解の集合から数値解を選択し，最適解を探す候補集合を作成する．

(S2) 候補集合の中から，妥当な制御器と妥当でないとはいえない制御器の集合を求める．

(S3) 求まった妥当な制御器と妥当でないとはいえない制御器に対して評価式 $K(\tilde{F})$ を計算し，最適な解を数値的最適制御器とする．

4.3 状態フィードバックの精度保証付き数値計算

本論文では，精度保証付き制御系設計手法の具体例として状態フィードバックを精度保証付きで設計する手法を提案する．

まず，状態フィードバックについて簡単に説明する．

システム

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

を考える．図 4.2 にこのシステムの状態フィードバックの概略図を示す．u は入力，F はフィードバックゲインである．図 4.2 のように状態 x をフィードバックすると

$$\begin{aligned} u &= -Fx \\ \dot{x} &= Ax + B(-Fx) = (A - BF)x \end{aligned}$$

となり，この解は

$$x(t) = e^{(A-BF)t}x_0$$

となる．ただし，x_0 は初期状態である．したがって，フィードバックゲイン F を操作することによって $A - BF$ の固有値を変え，閉ループ系の安定性を変更することが可能である．

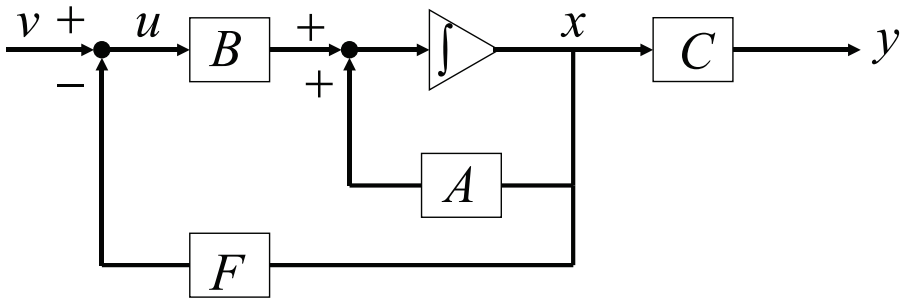

図 4.2: Schematic of state feedback

本研究では，この状態フィードバック制御を精度保証付きで行い，精度保証付き状態フィードバック制御系を設計することを考える．精度保証付き状態フィードバック制御系を設計するとは，設計時に発生する数値計算誤差の限界を計算し，設計仕様を含む制御器の集合を求めることで数値的品質を保証した制御系を設計することである．

以下の節では，状態フィードバック制御系設計の具体例として極配置問題と LQ 制御問題を考える．

4.4　極配置問題の検証付き解法

4.4.1　極配置問題

本節では，以下に示す極配置問題 [22, 23] を考える．極配置問題は (A, B) が可制御なシステム

$$\dot{x} = Ax + Bu : A \in \boldsymbol{R}^{n\times n}, B \in \boldsymbol{R}^{n\times m}$$

と指定極 P が与えられた時に $A - BF$ の固有値（極）を指定極 P に配置するフィードバックゲイン F を求める問題である．

4.4.2　極配置制御器検証問題

極配置問題 [22, 23] では，得られたフィードバックゲインを用いた閉ループ系の極が指定極に配置されているかどうかが重要である．問題によっては，閉ループ系の極が指定極と全く異なる位置に配置される可能性がある．そこで，極がどこに配置されるかの検証が必要である [13, 15]．よって，指定極が極配置制御器検証問題での制御器の満たすべき条件 C である．また，得られたフィードバックゲイン F を用いて $A - BF$ の固有値を精度保証付きで計算すると，結果が複素円区間として求まる．本問題では，制御器の保証できる性質（集合）S をこの円複素区間とする．もし，指定極がこの $A - BF$ の固有値の半径と中心からなる円盤上に存在すれば，フィードバックゲインによって得られた精度保証付き固有値は指定極を含む．これは，$A - BF$ の固有値が指定極と一致する可能性があることを示している．つまり，そのフィードバックゲインは妥当でないとはいえない制御器である．また，もし図 4.3(b) に示すように指定極が円盤上に存在しないならば，そのフィードバックゲインは妥当でない制御器である．図 4.3 に極配置制御器検証問題の概要を示す．

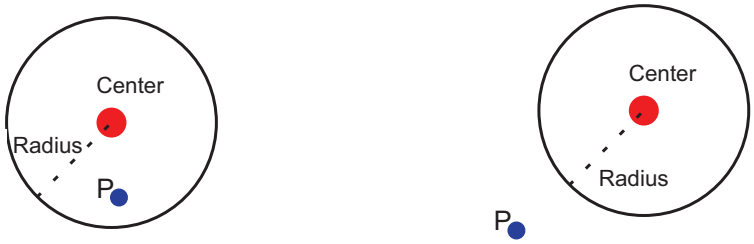

図 4.3: Schemeatic of Pole placement controller verification problem

4.4.3 数値的最適極配置問題

制御対象として可制御なシステム

$$\dot{x} = Ax + Bu$$

について考える．ただし，$x \in \boldsymbol{R}^n, u \in \boldsymbol{R}, A \in \boldsymbol{R}^{n\times n}, B \in \boldsymbol{R}^n$ であり，l 個の指定極があり，P_i が i 番目の指定極である極配置問題を精度保証付きで解いて得られたフィードバックゲインを $\mathcal{F}$ とする．このとき，数値的最適極配置問題はつぎのように与えられる [13, 15].

問題 3 (数値的最適極配置問題)

$$\min_{\tilde{F}\in\mathcal{F}} K(\tilde{F}) \tag{4.2}$$

$$subject\ to\ P_i \in [E]_i$$

ただし，

$$K(\tilde{F}) = \max_{1\leqq i\leqq l} \left|\left[\frac{[E]_i - P_i}{P_i}\right]\right| \tag{4.3}$$

$\tilde{F}$ は $\mathcal{F}$ 内の値を選択して作成した点行列（数値解），$[E]_i$ は i 番目の指定極 P_i に対応する $A - BF$ の固有値を精度保証付きで求めた複素円区間である．

極配置問題の理想的な状態は得られたフィードバックゲインによって配置された極が指摘極と完全に一致することなので，フィードバックゲインの評価に $E_i - P_i$ を用いる．(4.3) 式では $[E]_i - P_i$ を計算することで，区間 $[E]_i$ の中で P_i から最も遠い点までの距離を求め，評価を行っている．これにより，設計仕様を満たすこと（設計仕様に近い性能）を最も保証できる制御器（数値解）を区間行列の中から選択することができる．また，(4.3) 式の分母は i 番目の指定極である．極が虚

軸から遠い場合，状態の収束が早いので虚軸に近い極の方が重要である．そこで，原点で正規化することによって重み付けを行っている．そして，(4.3) 式を最小にするフィードバックゲインを数値的最適フィードバックゲインとする．

数値的最適極配置問題の解法手順を以下に示す．

(S1) 精度保証付きで求めたフィードバックゲインの集合から数値解を選択し，最適制御器を探す候補集合を作成する．

(S2) 候補集合の中から，妥当でないとはいえないフィードバックゲインの集合を求める．

(S3) 求まった妥当でないとはいえないフィードバックゲインに対して (4.3) 式を計算し，評価値を最小にするフィードバックゲインを数値的最適極配置制御器とする．

4.4.4　極配置問題の精度保証付き計算アルゴリズム

簡単化のために，1 入力系，つまり $m = 1$ の場合について考える．また，希望固有値を $r_0, r_1, \cdots, r_{n-1}$ とし，フィードバックゲイン $F = [f_0, f_1, \cdots, f_{n-1}]$ を求める．極配置問題を精度保証付きで解くアルゴリズムを以下に示す [13, 15]．

(S1) 可制御標準形への変換行列 T を区間演算を用いて，つぎのように求める．

$$T = \left[B, AB, A^2B, \cdots, A^{n-1}B\right] \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n-1} & 1 \\ \alpha_2 & \alpha_3 & \cdots & 1 & 0 \\ \vdots & \vdots & \iddots & 0 & \vdots \\ \alpha_{n-1} & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & \cdots & 0 \end{bmatrix}$$

ただし，$\alpha_0, \cdots, \alpha_{n-1}$ は特性多項式 $|sI - A| = s^n + \alpha_{n-1}s^{n-1} + \cdots + \alpha_1 s + \alpha_0$ の係数の下限と上限を CPU の丸めモード [35] を変更して求めたものである．

(S2) 状態フィードバックを施した閉ループ系の特性方程式は，

$$s^n + (\alpha_{n-1} + \bar{f}_{n-1})s^{n-1} + \cdots + (\alpha_1 + \bar{f}_1)s + (\alpha_0 + \bar{f}_0)s = 0$$

となる．一方，希望固有値をもつ特性方程式を

$$(s - r_0)(s - r_1)\cdots(s - r_{n-1}) = s^n + \theta_{n-1}s^{n-1} + \cdots + \theta_1 s + \theta_0$$

とする．この 2 式の係数を比較して，

$$\begin{cases} \theta_0 = \alpha_0 + \bar{f}_0 & \rightarrow & \bar{f}_0 = \theta_0 - \alpha_0 \\ \theta_1 = \alpha_1 + \bar{f}_1 & \rightarrow & \bar{f}_1 = \theta_1 - \alpha_1 \\ & \vdots & \\ \theta_{n-1} = \alpha_{n-1} + \bar{f}_{n-1} & \rightarrow & \bar{f}_{n-1} = \theta_{n-1} - \alpha_{n-1} \end{cases}$$

を区間演算で求める．

(S3) フィードバックゲイン F を

$$\begin{aligned} F = \bar{F}T^{-1} &= [\bar{f}_0,\ \bar{f}_1,\ \bar{f}_2,\ \cdots,\bar{f}_{n-1}]T^{-1} \\ &= [\theta_0 - \alpha_0,\ \theta_1 - \alpha_1,\ \cdots,\ \theta_{n-1} - \alpha_{n-1}]T^{-1} \end{aligned}$$

の区間行列演算によって求める．ただし，T^{-1} は線形方程式の精度保証を用いて T の逆行列（2.1 節参照）を計算する．

多入力系も同様に可制御標準形の変形から求めることができる [36].

4.5 LQ 制御問題の検証付き解法

4.5.1 LQ 制御問題

本節では，以下に示す LQ 制御問題 [22, 23] を考える．LQ 制御問題では (A, B) が可制御なシステム

$$\dot{x} = Ax + Bu : A \in \boldsymbol{R}^{n\times n}, B \in \boldsymbol{R}^{n\times m}$$

に対して，つぎの二次形式評価関数 (スカラ量) を考える．

$$J = \int_0^\infty \left\{x^T Q x + u^T R u\right\} dt$$

ここで，$Q \in \boldsymbol{R}^{n\times n}$, $R \in \boldsymbol{R}^{m\times n}$ は重み行列であり，Q は半正定，R は正定な対称行列である．LQ 制御問題はこの評価関数 J を最小にする最適フィードバック制御入力 u を求める問題である．この J を最小にする最適フィードバック制御入力 u_o はフィードバック係数行列 F を用いて，

$$\begin{aligned} u_o &= -Fx \\ &= -R^{-1}B^T P x \end{aligned}$$

と表せる．ただし，$F = R^{-1}B^TP$ である．また，P は任意の n 次の対称行列で，つぎのリカッチ方程式の唯一な正定値の解である．

$$A^TP + PA + Q - PBR^{-1}B^TP = 0 \tag{4.4}$$

4.5.2　LQ 制御器検証問題

LQ 制御器の検証条件として，以下のリカッチ方程式の数値解 $\tilde{P}$ に関する条件を用いる [16–18].

(C1) $\tilde{P} > 0$

(C2) $\tilde{P} = \tilde{P}^T$

(C3) $A^T\tilde{P} + \tilde{P}A + Q - \tilde{P}BR^{-1}B^T\tilde{P} = 0$

1 つ目の条件は固有値を精度保証付きで求め [32], その結果の領域が右半平面だけに存在するか（精度保証付き固有値の存在範囲が正の範囲だけか）を確かめることで確認できる．2 つ目の条件は対称な成分の値を比較することによって, 検証できる．3 つ目の条件は求めた解をリカッチ方程式の左辺に代入して区間演算を行い，残差ノルムが十分小さいかを確認する．ただし，ノルムの計算には文献 [9, 29] で定義されている無限大ノルムを用いる．

4.5.3　数値的最適 LQ 制御問題

制御対象として可制御なシステム

$$\dot{x} = Ax + Bu$$

について考える．ただし，$x \in \boldsymbol{R}^n, u \in \boldsymbol{R}, A \in \boldsymbol{R}^{n\times n}, B \in \boldsymbol{R}^n$ であり，A と B は可制御標準形で与えられているとし，重み行列 $Q \in \boldsymbol{R}^{n\times n},\ Q = Q^T \geqq 0,\ R = 1$ である LQ 制御問題のリカッチ方程式を精度保証付きで解いて得られた解を $\mathcal{P}$ とする．このとき，数値的最適 LQ 制御問題はつぎのように与えられる [16–18].

問題 4 (数値的最適 LQ 制御問題)

$$\min_{\tilde{P}\in\mathcal{P}} K(\tilde{P}) \tag{4.5}$$

subject to (C1), (C2), (C3)

ただし，

$$K(\tilde{P}) = \begin{cases} \max\limits_{1\leqq i\leqq n}\max\limits_{1\leqq j\leqq n}\left|\left[\dfrac{\left[\tilde{Q}\right]_{ij} - Q_{ij}}{Q_{ij}}\right]\right| \\ \qquad\qquad (Q_{ij}\neq 0) \\ \max\limits_{1\leqq i\leqq n}\max\limits_{1\leqq j\leqq n}\left|\left[\left[\tilde{Q}\right]_{ij}\right]\right| \\ \qquad\qquad (Q_{ij}= 0) \end{cases} \tag{4.6}$$

$\tilde{P}$ は $\mathcal{P}$ 内の値を選択して作成した点行列（数値解），Q_{ij} は Q の (i,j) 成分，$\left[\tilde{Q}\right]$ は

$$\tilde{Q} := -A^T\tilde{P} - \tilde{P}A + \tilde{P}BR^{-1}B^T\tilde{P}$$

を区間演算で計算して得られた区間行列である．

この問題で制御対象は可制御標準形であり $R = 1$ であるので，フィードバックゲイン F はリカッチ方程式の解 P の n 行目を取り出すだけで計算できる．そこで，数値的最適 LQ 制御問題ではフィードバックゲインの区間から数値解を選択する代わりに，リカッチ方程式の解の区間から最も設計仕様を満たすものを選択する．

また，$\tilde{P}$ がリカッチ方程式の真の解と一致するとき，$\tilde{Q}$ は Q と一致するので，$\tilde{P}$ の評価に $\tilde{Q} - Q$ を用いる．(4.6) 式では $\left[\tilde{Q}\right]_{ij} - Q_{ij}$ を計算することで区間 $\left[\tilde{Q}\right]_{ij}$ の中で Q_{ij} から最も遠い点までの距離を求め，評価を行っている．これにより，設計仕様を満たすこと（設計仕様に近い性能）を最も保証できる制御器（数値解）を区間行列の中から選択することができる．

数値的最適 LQ 制御問題のアルゴリズムを以下に示す．

(S1) 精度保証付き数値計算で求めたリカッチ方程式の解の集合から数値解を選択し，最適解を探す候補集合を作成する．

(S2) 候補集合の中から，妥当でないとはいえないリカッチ方程式の解の集合を求める．

(S3) 求まった妥当でないとはいえないリカッチ方程式の解に対して (4.6) 式を計算し，最適な解を求める．

(S4) 得られたリカッチ方程式の解からフィードバックゲインを求め，数値的最適制御器とする．

4.5.4　リカッチ方程式の精度保証付き計算アルゴリズム

精度保証付き数値計算を用いてリカッチ方程式を解く際に通常の近似計算と同じアルゴリズム（有本・ポッターの方法 [22]）をそのまま精度保証付きで行うと，精度の悪い解である大きな半径を持つ区間行列が求まる可能性がある．そこで，本研究では解の精度を向上させるためにリカッチ方程式を非線形方程式としてとらえ，固有値の精度保証計算を用いて求まったリカッチ方程式の解を初期区間としてクラフチック法 [4–6] を適用する手法を提案する [16–18]．クラフチック法を用いたリカッチ方程式の精度保証付き計算アルゴリズムを以下に示す．

(S1) ハミルトン行列

$$H = \begin{bmatrix} A & -BR^{-1}B^T \\ -Q & -A^T \end{bmatrix}$$

の固有値を精度保証付き数値計算で求める [32]．

(S2) 固有値の中で負の実部を持つ（精度保証付き固有値の存在範囲が左半平面にある）ものを求めて，$r_1, r_2, \cdots, r_n$ とする．

(S3) $r_1, r_2, \cdots, r_n$ に対応する固有空間の基底を精度保証付き数値計算で求め [32]，つぎのように二つに分ける．

$$\begin{bmatrix} v_1 \\ u_1 \end{bmatrix}, \cdots, \begin{bmatrix} v_n \\ u_n \end{bmatrix}$$

ただし，$v_i \in I\boldsymbol{C}^{n\times 1}, u_i \in I\boldsymbol{C}^{n\times 1}$ である．

(S4) リカッチ方程式の解 P をつぎの区間演算により求める．

$$P = [u_1, u_2, \cdots, u_n][v_1, v_2, \cdots, v_n]^{-1}$$

(S5) リカッチ方程式を非線形方程式 $f(P) = A^TP + PA + Q - PBR^{-1}B^TP = 0$ として定義し，求めたリカッチ方程式の解 P を初期区間としてクラフチック法を適用する．

上記のアルゴリズムにより，リカッチ方程式の真の解を含む半径の小さい区間行列を求めることができる．また，本提案手法は非線形方程式を解く前の段階ですでに（固有値に基づく計算により）LQ 制御器に必要な条件とクラフチック法の

初期条件を満たしていることが期待できる初期解区間が求まっており，微分値は自動微分 [4, 5, 30] を用いて計算を行うという特徴を持つ.

上記のリカッチ方程式の精度保証付き計算アルゴリズムにより P の区間行列を求めた後は，$F = R^{-1}B^T P$ を区間演算を用いて計算することにより F を精度保証付きで求めることができる.

4.6 数値例

4.6.1 極配置問題の精度保証付き数値計算

数値例として以下のシステムを考える．指定極は $P = -1, -5, -10, -15, -20, -25$ である.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50000 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

この問題は A の条件数が 50000 であり，非常に感度の高い問題である.

まず，通常の近似計算でフィードバックゲインを求める．計算には，Java 数値計算パッケージ NFC[37] を用いた（文献 [23] に示されている可制御標準形からの方法を用いている. 特別な最適化は行っていない）.

```
===近似解 F(1 x 6) Matrix===
                (  1)                          (  2)
(  1) -9.639806218498292000E-01  8.509152983176330000E+00
                (  3)                          (  4)
(  1) -2.418811769229065300E+01  2.785936334204669000E+01
                (  5)                          (  6)
(  1) -1.128626546904882800E+01  5.060212104148602000E+04
```

このフィードバックゲインの近似値を用いた $A - BF$ の固有値を以下に示す.

```
=== 固有値 (6 x 1) CoMatrix ===
        [ (  1)-Real ]                  [ (  1)-Imag ]
(  1) 5.5565146900200535 00E+00  1.0158624986104368 00E+01
(  2) 5.5565146900200535 00E+00 -1.0158624986104368 00E+01
(  3)-4.7366788701348830 00E-01  0.0000000000000000 00E+00
(  4)-6.4304072501859990 00E-01  2.8107122195741603 00E+00
(  5)-6.4304072501859990 00E-01 -2.8107122195741603 00E+00
(  6)-5.9640447407103990 00E+02  0.0000000000000000 00E+00
```

上記の結果は指定極と全く対応していない．よって，上記の近似的に求めたフィードバックゲインは妥当でない．

そこで，上記の問題を精度保証付きで解く．

この問題についてフィードバックゲインを求めると，フィードバックゲインの区間は以下のようになる．

```
 ===  下限  (   1  x   6)  Matrix  ===
             (  1)                          (  2)
( 1) -9.6097921958605760 00E-01  8.4826593063552980 00E+00
             (  3)                          (  4)
( 1) -2.4112806768490472 00E+01  2.7772621809645454 00E+01
             (  5)                          (  6)
( 1) -1.1251125112559482 00E+01  5.0091069629984300 00E+04
 ===  上限  (   1  x   6)  Matrix  ===
             (  1)                          (  2)
( 1) -9.6097921958276670 00E-01  8.4826593063890330 00E+00
             (  3)                          (  4)
( 1) -2.4112806768321953 00E+01  2.7772621809843788 00E+01
             (  5)                          (  6)
( 1) -1.1251125112462898 00E+01  5.0091069629984500 00E+04
```

この区間の中に，真の解が含まれていることが保証されている．

4.6.2　LQ 制御問題の精度保証付き数値計算

LQ 制御問題の精度保証付き数値計算の数値例としてつぎの問題，文献 [38] の Example 12 を考える．制御対象の係数行列と重み行列を

$$A = VA_0V, \quad B = I, \quad Q = VQ_0V, \quad R = \varepsilon I$$

とする．ただし，

$$A_0 = \varepsilon\,\mathrm{diag}\,(1,2,3)\,, \quad Q_0 = \mathrm{diag}\left(\frac{1}{\varepsilon},1,\varepsilon\right)$$

$$V = I - \frac{2}{3}vv^T, \quad v^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

であり，ε はパラメータである．

この問題は解析的に解くことが可能であり，リカッチ方程式の解およびそのときのフィードバックゲインは

$$P = V\,\mathrm{diag}(x_1, x_2, x_3)\,V \tag{4.7}$$

$$F = \frac{1}{\varepsilon}\,V\,\mathrm{diag}(x_1, x_2, x_3)\,V$$

である．ただし，

$$\begin{aligned} x_1 &= \varepsilon^2 + \sqrt{\varepsilon^4 + 1} \\ x_2 &= 2\varepsilon^2 + \sqrt{4\varepsilon^4 + \varepsilon} \\ x_3 &= 3\varepsilon^2 + \sqrt{9\varepsilon^4 + \varepsilon^2} \end{aligned}$$

である．以下では ϵ の値を $\epsilon = 10^6$ とする．

まず，精度保証付き数値計算と多倍長計算 [20] を組み合わせて計算した (4.7) 式の値の小数点以下 20 桁までの値を示す．

$$P_{ans} = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_2 & p_4 & p_5 \\ p_3 & p_5 & p_6 \end{bmatrix} \tag{4.8}$$

ただし，

$$\begin{aligned} p_1 &= 4.66666666666674074085 \times 10^{12} \\ p_2 &= 1.33333333333340740735 \times 10^{12} \\ p_3 &= -3.70369259260360082304 \times 10^{-2} \\ p_4 &= 4.00000000000007407410 \times 10^{12} \\ p_5 &= -1.33333333333337037042 \times 10^{12} \\ p_6 &= 3.33333333333335185196 \times 10^{12} \end{aligned}$$

である．この結果は 10 進数 100 桁分の精度で計算したものであり，区間半径も約 100 桁である．つまり，この 21 桁の値は完全に正しいことが保証されている．

つぎに Matlab[34] のリカッチ方程式のソルバーを用いて，リカッチ方程式の解とフィードバックゲインを求める．そのときの値は，

```
P =
  Columns 1 through 2
    4.666597159042693e+012   1.333574498305712e+012
    1.333574498305712e+012   4.000426567390165e+012
   -4.985213423937625e+008  -1.333624105707470e+012
  Column 3
   -4.985213423937625e+008
   -1.333624105707470e+012
    3.333446698627802e+012
F =
  Columns 1 through 2
    4.666597159042694e+006   1.333574498305712e+006
    1.333574498305712e+006   4.000426567390165e+006
   -4.985213423937625e+002  -1.333624105707470e+006
  Column 3
   -4.985213423937625e+002
   -1.333624105707470e+006
    3.333446698627802e+006
```

ただし Matlab のバージョンは 7.5.0(R2007b) であり，リカッチ方程式のソルバーは care である．エラーメッセージや警告は出ていないが，P の $(1,3)$ 成分と $(3,1)$ 成分が (4.8) 式と大きく異なっていることがわかる．この P を (4.4) 式の左辺に代入して区間演算を行い残差ノルムを求めると，

$$5.790255989639168 \times 10^{15}$$

となり，非常に大きな値が求まる．つまり，この P は妥当でない．

そこで，この問題を本論文で提案した手法を Java で実装したプログラムを用いて精度保証付きで解くと，リカッチ方程式の解は

```
===  IP 中心  (   3  x   3)  Matrix  ===
                (  1)                          (  2)
(1)  4.666666666666742000000e+12  1.333333333333407700000e+12
(2)  1.333333333333407700000e+12  4.000000000000075000000e+12
(3) -3.746610445601849400000e-02 -1.333333333333369400000e+12
                (  3)
(1) -3.746729716435182000000e-02
(2) -1.333333333333369400000e+12
(3)  3.333333333333353000000e+12
===  IP 半径  (   3  x   3)  Matrix  ===
                (  1)                          (  2)
(1)  1.953125000000000000000e-03  1.953125000000000000000e-03
(2)  1.708984375000000000000e-03  2.929687500000000000000e-03
(3)  6.600164609057435000000e-04  1.953125000000000000000e-03
                (  3)
(1)  6.185349794243372000000e-04
(2)  1.708984375000000000000e-03
(3)  1.953125000000000000000e-03
```

となる．この区間にリカッチ方程式の解，(4.8) 式が存在することが保証されてい

る．この区間の中心の有効桁数を求めると，

```
===  有効桁数  (   3  x   3)  IntegerMatrix  ===
                (  1)       (  2)       (  3)
 (  1)           15          14           1
 (  2)           14          15          14
 (  3)            1          14          15
```

となる．また，求めた区間 IP の中心を (4.4) 式の左辺に代入して残差ノルムを求めると，

$$1.1648 \times 10^5$$

となる．

このように，精度保証付き数値計算で求まった結果は中心と半径で得られるため，その結果から中心の値を用いるかどうかを選択できる．また，残差ノルムが小さい時には得られた結果の誤差も小さく，残差ノルムが大きい時には得られた結果の誤差も大きくなる．これより，求まったリカッチ方程式の解を (4.4) 式の左辺に代入して残差ノルムを求める手法の妥当性がわかる．

リカッチ方程式の解を精度保証付きで求めるのにかかった時間は，0.437 秒である．ただし計算機環境は，OS: Windows XP Professional SP3，CPU: Core2Duo E6750(2.66GHz)，メモリ: 2GB である．

4.6.3 数値的最適 LQ 制御問題

つぎに数値的最適 LQ 制御問題の数値例としてつぎの問題，文献 [38] の Example 9 を考える．

制御対象の係数行列と重み行列を

$$A = \begin{bmatrix} 0 & \varepsilon \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = 1$$

とする．ただし，ε はパラメータである．この問題は解析的に解くことが可能であり，リカッチ方程式の解およびそのときのフィードバックゲインは

$$P = \begin{bmatrix} \dfrac{\sqrt{1+2\epsilon}}{\epsilon} & 1 \\ 1 & \sqrt{1+2\epsilon} \end{bmatrix} \tag{4.9}$$

$$F = \begin{bmatrix} 1 & \sqrt{1+2\epsilon} \end{bmatrix}$$

である．以下では ϵ の値を $\epsilon = 10^{-15}$ とする．

まず，精度保証付き数値計算と多倍長計算を組み合わせて計算した (4.9) 式の値の浮動小数点以下 17 桁までの値を示す．

$$P_{ans} = \begin{bmatrix} 1.00000000000000099 \times 10^{15} & 1.00000000000000000 \\ 1.00000000000000000 & 1.00000000000000099 \end{bmatrix} \tag{4.10}$$

この結果は 10 進数 100 桁分の精度で計算したものであり，区間半径も約 100 桁である．つまり，この 18 桁の値は完全に正しいことが保証されている．

つぎに Matlab のリカッチ方程式のソルバーを用いて，リカッチ方程式の解とフィードバックゲインを求める．そのときの値は，

```
P =
    9.999998393676155e+014   1.000000000000013e+000
    1.000000000000013e+000   1.000000000000001e+000
F =
    1.000000000000013e+000   1.000000000000001e+000
```

ただし Matlab のバージョンは 7.5.0(R2007b) であり，リカッチ方程式のソルバーは care である．この P を (4.4) 式の左辺に代入して区間演算を行い残差ノルムを求めると，

$$1.606324260272629 \times 10^{-7}$$

となる．

つぎに，この問題について数値的最適 LQ 制御問題を解く．

(S1) この問題のリカッチ方程式の解を精度保証付きで求めると，

リカッチ方程式の解の区間

```
===  IP 下限  (   2  x   2)  Matrix  ===
                  (  1)                          (  2)
(  1) 1.000000000000000060000e+15 9.999999999999999600000e-01
(  2) 9.999999999999999600000e-01 1.000000000000000090000e+00
===  IP 上限  (   2  x   2)  Matrix  ===
                  (  1)                          (  2)
(  1) 1.000000000000000160000e+15 1.000000000000000040000e+00
(  2) 1.000000000000000040000e+00 1.000000000000000130000e+00
```

となる．この区間にリカッチ方程式の解，(4.10) 式が存在することが保証されている．求まった区間行列 IP の $(1,1)$ 成分の区間には 9 個，$(1,2)$ 成分の区間には 7 個，$(2,1)$ 成分の区間には 7 個，$(2,2)$ 成分の区間には 3 個の浮動小数点がある．したがって，候補集合の要素は $9 \times 7 \times 3 = 189$ 個である．

(S2) 189 個の候補の内，妥当でないとはいえないリカッチ方程式の解の個数は 23 個であった．

(S3) これらのリカッチ方程式の解について (4.6) 式を計算する．(4.6) 式を最も小さくする解は

$$P = \begin{bmatrix} 1.0000000000000006 \times 10^{15} & 1.0000000000000000 \\ 1.0000000000000000 & 1.0000000000000009 \end{bmatrix}$$

であった．そのときの (4.6) 式の値は，

$$8.88178419700125616 \times 10^{-16}$$

である．
(S4) 求まったリカッチ方程式の解からフィードバックゲインを求めると，

$$F = \begin{bmatrix} 1.0000000000000000 & 1.0000000000000009 \end{bmatrix}$$

となる．
上記の P をリカッチ方程式の左辺に代入し残差ノルムを求めると，

$$1.085394 \times 10^{-15}$$

となり，非常に小さい値が求まっている．
今回の問題を解くのにかかった時間を表 4.1 に示す．計算機環境は，前記の環境と同じである．この計算全体にかかる時間が約 0.7 秒であるので実用的な時間で求めることができている．

表 4.1: Computational Time

Step	Time(ms)
Step (S1)(Computation of interval P + select candidates of optimal P)	171.0(156.0+15.0)
Step (S2)	547.0
Step (S3)	16.0
Step (S4)	0.0
Total	734.0

なお、クラフチック法を適用しなかった場合にかかる時間を表 4.2 に示す．クラフチック法を適用することにより約 1/3 の時間で問題を解くことができており，提案したリカッチ方程式の解法の有効性がわかる．

表 4.2: Computational Time without Krawczyk method

Step	Time(ms)
Step (S1)(Computation of interval P + select candidates of optimal P)	141.0(94.0+47.0)
Step (S2)	2078.0
Step (S3)	16.0
Step (S4)	0.0
Total	2235.0

第5章　数値的最適制御器問題の効率的解法

数値的最適制御器の候補集合を作成するためには，区間行列（制御器の集合）の要素の区間から浮動小数点を選択する必要がある．この時，精度保証付きで求まった解の精度が悪いと大きな半径を持つ区間行列（制御器の集合）が求まる．半径の大きな区間行列が求まると，区間内の浮動小数点の個数が増えてしまう．そのため数値的最適制御器の候補が多くなってしまい，数値的最適制御器の選択に多くの時間が必要となる．

この問題を解決するために，以下の 2 つの方法を提案する．

1. 区間行列に修正は行わず， 部分探索を行う

2. 区間行列に修正を行った後に，全探索を行う

前者の方法では，ヒューリスティックの一種である遺伝的アルゴリズム（GA）[26] を用いることで候補集合を作成する [15]. 後者の方法ではクラフチック法 [4–6] や多倍長演算 [19, 20] を用いて区間の半径を小さくした後に全探索を行う [17, 18, 27, 28].

5.1　GA を用いた候補集合の作成

本節では数値的最適制御問題において候補者集合を作成する際に遺伝的アルゴリズム（GA）[26] を用いる手法を提案する [15]. 図 5.1 に GA の計算の流れを示す．

本論文では GA の一種である実数値 GA を用いる．通常の GA では固体をバイナリ文字列として表現する手法がよく用いられる．これにたいして，実数値 GA では実数値をそのまま固体として扱うことができる．そこで本研究ではこの実数値 GA を用い，個体を区間内の浮動少数点のインデックスの行列とし，計算を行う際にはそのインデックスの点の表す値を用いる．そして，数値的最適制御問題の評価関数を適応度の計算に用い，交叉に UNDX 法を使う，そして突然変異には一様乱数を用いる．GA を用いることで，区間行列から候補集合を効率的に選択することができる．

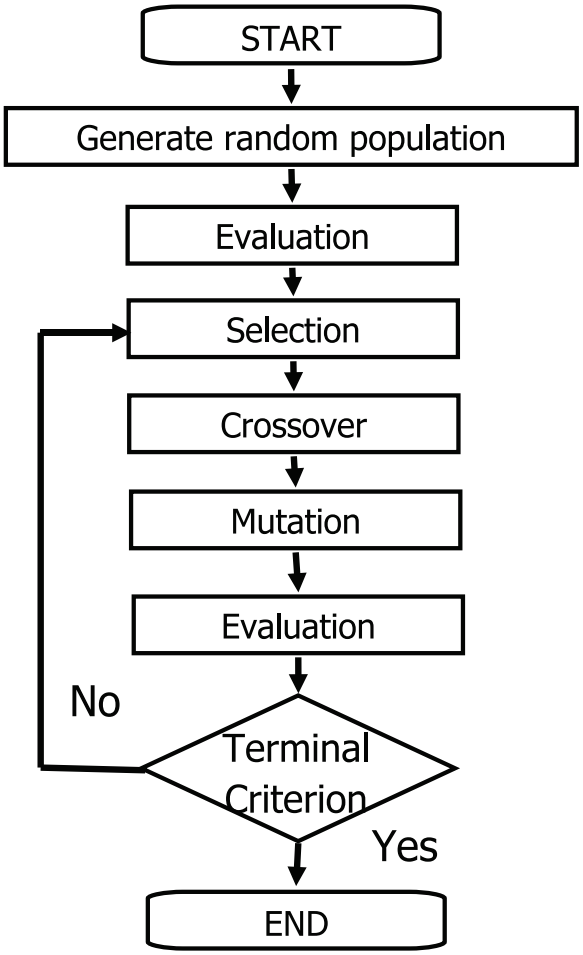

図 5.1: Flow of GA

UNDX(Unicode Normal Distribution Crossover) 法は，実数を個体として用いる実数値 GA の一種である．両親を a, b とする．そして，次の個体 c を以下の式で決定する．

$$c = m + \xi d$$

ただし，$m = (a+b)/2$, $d = a - b$, $\xi \sim N(0, \sigma^2)$ であり，σ はパラメータである．

また，一様乱数による突然変異では固体 x が区間 $[X_{MIN}, X_{MAX}]$ 内に存在するとする．まず，方向（正/負）を確率 $1/2$ で選ぶ．そして，方向が正であれば区間 $[x, \min(x+M, X_{MAX})]$ から，負であれば区間 $[\min(X_{MIN}, x-M), x]$ の中から一様乱数で選択する．

5.2 多倍長演算を用いた精度の向上

上記の GA を用いる方法は局所最適解しか求まらないという問題がある．そこで，区間の精度を向上させた後に全探索を行う手法を提案する．この方法は全探索を行うので，大域的最適解が求まる．具体的手法としては，クラフチック法を用いて解の精度を向上させることができる [16, 17]．しかし，クラフチック法を用いる方法は，クラフチック法が適用できるように初期区間行列の精度が良い必要がある．また，初期区間行列の精度の問題でクラフチック法が適用できなかった場合や,（倍精度の精度の限界により）クラフチック法を適用しても区間幅が大きいままの（難しい）問題では，候補の数を減少させることができない．

精度に関する問題を解決し，精度を向上させる手法として多倍長演算を用いる手法 [19, 20] が提案されている．また，多倍長演算を精度保証付き数値計算に適用する手法 [21] も提案されている．多倍長演算を用いることで，通常の計算で用いられる倍精度演算よりも高精度に計算が行えるため，非常に小さい半径を持つ区間行列を計算することができる．

本節では半径の小さい区間行列を求めることで数値的最適制御器の候補を減少させるために，区間の精度を多倍長演算を用いて改善した後に倍精度の区間行列に近似する手法を提案する [27, 28]．

本提案手法ではまず多倍長演算と精度保証付き数値計算を用いて，区間行列を求める．この時，通常の倍精度による近似計算では不可能な高精度の区間行列が求まる．次に倍精度の数値解を求めるために，図 5.2 に示すように多倍長の区間の端点を倍精度に丸める．この時，丸めモードの変更も行い，区間の中に真の解を含む倍精度の区間を求めるようにしている．図 5.2 に示すように，多倍長精度区間の中に倍精度浮動小数点が含まれない場合，得られる倍精度区間の中には 2 個の浮動小数点のみが含まれる．また，多倍長精度区間の中に倍精度浮動小数点が含まれる場合，得られる倍精度区間の中には 3 個の浮動小数点のみが含まれる．よって，結果として得られる倍精度の区間の中には，2 個もしくは 3 個の倍精度浮動小数点数のみが含まれる．

本提案手法を用いることにより，区間行列内の浮動小数点を大幅に減らすことができ，全探索が高速に行えるようになる．

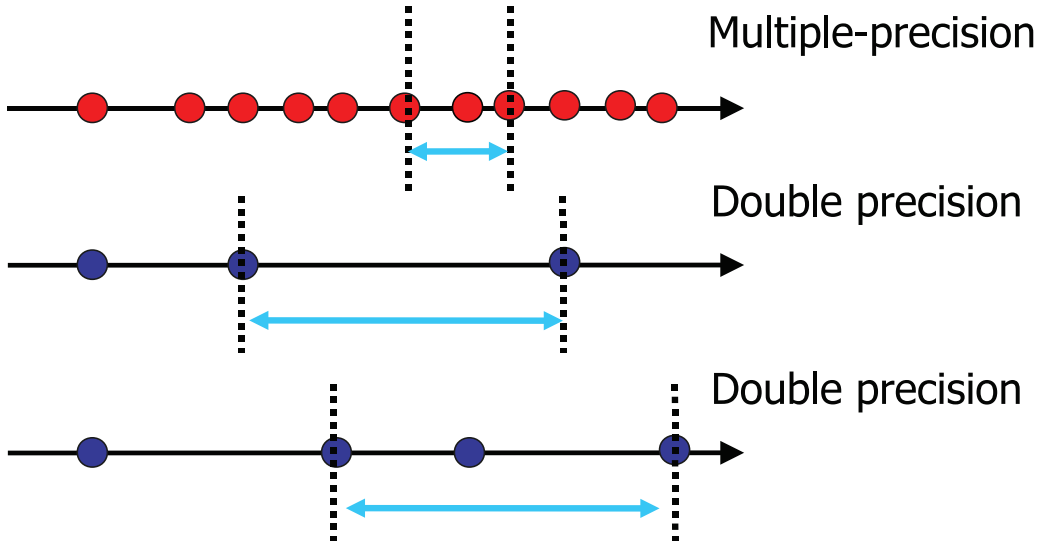

図 5.2: Approximate Multiple-precision to double precision

5.3　数値例

5.3.1　GA を用いた数値的最適極配置問題

数値的最適極配置問題の数値例として，以下のシステムを考える．

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50000 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

また，指定極を $P = -1, -5, -10, -15, -20, -25$ とする．この問題は A の条件数が 50000 であり，非常に感度の高い問題である．

まず，通常の近似計算でフィードバックゲインを求める．計算には，Java 数値計算パッケージ NFC[37] を用いた（文献 [23] に示されている可制御標準形からの方法を用いている．特別な最適化は行っていない）．

```
===近似解 F(1 x 6) Matrix===
             (  1)                        (  2)
( 1) -9.639806218498292000E-01  8.509152983176330000E+00
             (  3)                        (  4)
( 1) -2.418811769229065300E+01  2.785936334204669000E+01
             (  5)                        (  6)
( 1) -1.128626546904882800E+01  5.060212104148602000E+04
```

このフィードバックゲインの近似値を用いた $A - BF$ の固有値を以下に示す.

```
=== 固有値 (6 x 1) CoMatrix ===
        [ (  1)-Real ]                  [ (  1)-Imag ]
(  1) 5.556514690020053500E+00  1.015862498610436800E+01
(  2) 5.556514690020053500E+00 -1.015862498610436800E+01
(  3)-4.736678870134883000E-01  0.000000000000000000E+00
(  4)-6.430407250185999000E-01  2.810712219574160300E+00
(  5)-6.430407250185999000E-01 -2.810712219574160300E+00
(  6)-5.964044740710399000E+02  0.000000000000000000E+00
```

上記の結果は指定極と全く対応していない．よって，上記の近似的に求めたフィードバックゲインは妥当でない.

そこで，上記の問題について数値的最適極配置問題を解く.

(S1) この問題についてフィードバックゲインを求めると，フィードバックゲインの区間は以下のようになる.

```
 ===  下限  (   1  x   6)  Matrix  ===
             (  1)                        (  2)
( 1) -9.609792195860576000E-01  8.482659306355298000E+00
             (  3)                        (  4)
( 1) -2.411280676849047200E+01  2.777262180964545400E+01
             (  5)                        (  6)
( 1)  1.125112511255948200E+01  5.009106962998430000E+04
 ===  上限  (   1  x   6)  Matrix  ===
             (  1)                        (  2)
( 1) -9.609792195827667000E-01  8.482659306389033000E+00
             (  3)                        (  4)
( 1) -2.411280676832195300E+01  2.777262180984378800E+01
             (  5)                        (  6)
( 1) -1.125112511246289800E+01  5.009106962998450000E+04
```

この区間の中心の有効桁数を求めると,

```
===  有効桁数  (   1  x   6)  IntegerMatrix  ===
             (  1)       (  2)       (  3)
 (  1)         12          12          11
             (  4)       (  5)       (  6)
 (  1)         11          11          14
```

となる．また，上記の区間内に含まれる浮動小数点の個数を表 5.1 に示す．

表 5.1: Number of floating point in interval

Element	Number of floating point
(1,1)	29463
(1,2)	18992
(1,3)	47435
(1,4)	55827
(1,5)	54373
(1,6)	29

表 5.1 より，上記の区間で全探索を行うと候補集合の要素数は，

$$29463 \times 18992 \times 47435 \times 55827 \times 54373 \times 29 = 1518553680$$

となる．試しに 1000 個の候補に関して計算を行ったところ，9.712[s] かかった．そのため，約 15 億個の候補に対して計算を行うと約 170 日かかることが分かる．このため，この問題を全探索で解くことは現実的ではない．

そこで，GA を用いて候補集合を作成する．用いた GA のパラメータを表 5.2 に示す．

表 5.2: Parameters of GA

Parameter	value
Population size	60
Number of generation	100
Number of elite	10
Crossover rate	100%
σ	0.5
Mutation rate	25%
M	1/10 of numbers of floating point of interval

パラメータとして集団の個体数を 60，世代交代の数を 100，エリート数を 10，交叉率を 100%，σ の値を 0.5，突然変異率を 25%，M の値を区間内の浮動小数点数の個数の $\frac{1}{10}$ とした．そして，求めたフィードバックゲインに対して，$A - BF$ の精度保証付き固有値を計算する．

(S2) 妥当でないとはいえないフィードバックゲインの個数は 7 個であった.
(S3) (4.3) 式に基づいて，7 個のフィードバックゲインの評価値を求めた．(4.3) 式を最小にするフィードバックゲインは以下の値であった.

```
 ===  F  (   1  x   6)  Matrix  ===
              (  1)                        (  2)
( 1) -9.609792195844106000E-01  8.482659306372607000E+00
              (  3)                        (  4)
( 1) -2.411280676840988600E+01  2.777262180975247000E+01
              (  5)                        (  6)
( 1) -1.125112511251553000E+01  5.009106962998439000E+04
```

そのときの (4.3) 式の値は

$$5.432948863478507 \times 10^{-10}$$

であった.

数値的最適フィードバックゲインを用いたときの $A - BF$ の精度保証付き固有値は以下の円盤上に存在する.

```
=== 円盤（固有値の存在範囲）中心 (6 x 1) CoMatrix ===
         [ (  1)-Real ]         [(  1)-Imag ]
(  1)-1.000000000002854600E+00 0.000000000000000000E+00
(  2)-5.000000000033347000E+00 0.000000000000000000E+00
(  3)-9.999999997785450000E+00 0.000000000000000000E+00
(  4)-1.500000000168764200E+01 0.000000000000000000E+00
(  5)-1.999999999991897000E+01 0.000000000000000000E+00
(  6)-2.499999999954850000E+01 0.000000000000000000E+00
=== 円盤（固有値の存在範囲）半径  (6 X 1) Matrix ===
              (  1)
(  1)  7.627531467764463000E-12
(  2)  3.508607539019007000E-10
(  3)  3.432007441453436400E-09
(  4)  6.461781269090835500E-09
(  5)  1.066656583505956800E-08
(  6)  4.605143259111369500E-09
```

上記の固有値の実部の区間の下限と上限は

```
=== Infimum of eigen value(6 x 1) Matrix ===
             (  1)
(  1) -1.0000000000010482300E+00
(  2) -5.0000000003842080000E+00
(  3) -1.0000000003210554000E+01
(  4) -1.5000000008149424000E+01
(  5) -2.0000000010585538000E+01
(  6) -2.5000000004153645000E+01
=== Supremum of eigen value(6 x 1) Matrix ===
             (  1)
(  1) -9.9999999995227000E-01
(  2) -4.9999999996824850000E+00
(  3) -9.9999999963465370000E+00
(  4) -1.4999999995225860000E+01
(  5) -1.9999999989252400000E+01
(  6) -2.4999999994943355000E+01
```

となる．上記の結果は，$A - BF$ の固有値の区間が指定極を含んでいることを示している．

この問題を解くのにかかった時間を表 5.3 に示す．計算環境は OS: Windows XP Professional, CPU: Core2Duo E6750(2.66GHz),Memory: 2GB である．

表 5.3: Time to compute example using GA

Step	Time(ms)
Step(S1)(Computation of interval F + select candidate of optimal F)	99813(188+99625)
Step(S2)(Verified pole assignment method)	562
Step(S3)(Evaluation of optimal feedback gain with guarantee of quality F)	0
Total	100375

5.3.2　多倍長演算を用いた数値的最適極配置問題

多倍長演算を用いた数値的最適極配置問題の例として，上記の GA と同じ数値例を用いる．

(S1) 多倍長演算と精度保証付き数値計算を用いて求めた，フィードバックゲインの区間行列の中心と半径を以下に示す．ただし，多倍長浮動小数点数として 10 進数で 115 桁を用いている．

```
===  中心  (   1  x   6)  NumericalMatrix  ===
              (  1)                    (  2)
(  1) -9.6097921958439169e-1  8.4826593063722549e+0
              (  3)                    (  4)
(  1) -2.4112806768406104e+1  2.7772621809744780e+1
              (  5)                    (  6)
(  1) -1.1251125112511251e+1  5.0091069629984385e+4
===  半径  (   1  x   6)  NumericalMatrix  ===
              (  1)                    (  2)
(1) 2.5865307337910573e-111 3.2080396965057747e-110
              (  3)                    (  4)
(1) 1.1250066755244619e-109 1.4784749717980267e-109
              (  5)                    (  6)
(1) 6.9001968743631842e-110 1.9959186750038206e-110
```

スペースの都合から，初めの 17 桁のみが表示されている．多倍長演算を用いることで，通常の倍精度では不可能な高精度な解を求めることができている．多倍長演算を用いて求めた区間行列の半径の最大値は

$$1.4784749717980267 \times 10^{-109}$$

である．上記の区間行列を倍精度に近似すると，

```
===  中心  (   1  x   6)  NumericalMatrix  ===
              (  1)                    (  2)
(  1) -9.6097921958439170e-01 8.4826593063722560e+00
              (  3)                    (  4)
(  1) -2.4112806768406106e+01 2.7772621809744780e+01
              (  5)                    (  6)
(  1) -1.1251125112511252e+01 5.0091069629984384e+04
 ===  半径  (   1  x   6)  Matrix  ===
              (  1)                    (  2)
(  1)  1.1102230246251565e-16 1.7763568394002505e-15
              (  3)                    (  4)
(  1)  3.5527136788005010e-15 3.5527136788005010e-15
              (  5)                    (  6)
(  1)  1.7763568394002505e-15 7.2759576141834260e-12
```

となる．上記の区間行列の半径の最大値は，

$$7.2759576141834260 \times 10^{-12}$$

である．これは通常の倍精度演算のみを用いて求めた値である

$$1.0186340659856796 \times 10^{-10}$$

よりも小さい．

(S2) 倍精度に近似した行列の区間内を全探索し，候補集合を作成した．妥当でないとはいえない制御器を求めると，その個数は $729 = 3^6$ であり全ての候補が妥当でないとはいえないと判定された．
(S3) (4.3) 式の最小値は

$$4.1918204764994734 \times 10^{-10}$$

であり，前節の GA による結果 ($5.378141904136705 \times 10^{-10}$) よりも良い結果が得られた．また，そのときのフィードバックゲインの値は

```
===  F  (   1  x   6)  NumericalMatrix  ===
              (  1)                    (  2)
(  1) -9.6097921958439160e-1  8.4826593063722560e+0
              (  3)                    (  4)
(  1) -2.4112806768406102e+1  2.7772621809744780e+1
              (  5)                    (  6)
(  1) -1.1251125112511254e+1  5.0091069629984384e+4
```

であった．

この問題を解くのにかかった時間を表 5.4 に示す．ただし，計算環境は OS: Windows XP Professional，CPU: Intel Core2Duo E6750(2.66GHz)，Memory:2GB である．表 5.3 の GA の結果と比較すると，より短い時間で最適解が求まっていることが分かる．

表 5.4: Time to compute example using Mlutiple-precision

Step	Time(ms)
Step(S1)(Computation of interval F + select candidate of optimal F)	778.1(770.3+7.8)
Step(S2)(Verified pole assignment method)	8410.7
Step(S3)(Evaluation of optimal feedback gain with guarantee of quality F)	83.1
Total	9271.9

5.3.3 多倍長演算を用いた数値的最適 LQ 制御問題

数値例としてつぎの問題，文献 [38] の Example 12 を考える．制御対象の係数行列と重み行列を

$$A = VA_0V, \quad B = I, \quad Q = VQ_0V, \quad R = \varepsilon I$$

とする．ただし，

$$A_0 = \varepsilon\,\mathrm{diag}\,(1,2,3)\,, \quad Q_0 = \mathrm{diag}\left(\frac{1}{\varepsilon},1,\varepsilon\right)$$

$$V = I - \frac{2}{3}vv^T, \quad v^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

であり，ε はパラメータである．以下では ϵ の値を $\epsilon = 10^6$ とする．

まず，この問題を精度保証付き（倍精度）で解くと，リカッチ方程式の解は

```
===  IP中心  (   3  x   3)  Matrix  ===
                (  1)                          (  2)
(1)  4.666666666666742000000e+12  1.333333333333407700000e+12
(2)  1.333333333333407700000e+12  4.000000000000075000000e+12
(3) -3.746610445601849400000e-02 -1.333333333333369400000e+12
                (  3)
(1) -3.746729716435182000000e-02
(2) -1.333333333333369400000e+12
(3)  3.333333333333353000000e+12
===  IP半径  (   3  x   3)  Matrix  ===
                (  1)                          (  2)
(1)  1.953125000000000000000e-03  1.953125000000000000000e-03
(2)  1.708984375000000000000e-03  2.929687500000000000000e-03
(3)  6.600164609057435000000e-04  1.953125000000000000000e-03
                (  3)
(1)  6.185349794243372000000e-04
(2)  1.708984375000000000000e-03
(3)  1.953125000000000000000e-03
```

となる．この区間にリカッチ方程式の解が存在することが保証されている．この区間の中心の有効桁数を求めると，

```
===  有効桁数  (   3  x   3)  IntegerMatrix  ===
              (  1)       (  2)       (  3)
 (  1)          15          14           1
 (  2)          14          15          14
 (  3)           1          14          15
```

となる．また，求めた区間 IP の中心を (4.4) 式の左辺に代入して残差ノルムを求めると，

$$1.1648 \times 10^5$$

表 5.5: Number of floating point in interval

Element	Number of floating point
(1,1)	7
(1,2)	19
(1,3)	18446517884743946444
(2,1)	17
(2,2)	13
(2,3)	17
(3,1)	18446520139345999918
(3,2)	19
(3,3)	11

となる．上記の区間内の浮動小数点の個数を表 5.5 に示す．

リカッチ方程式の解を精度保証付きで求めるのにかかった時間は，0.437 秒である．ただし計算機環境は，OS: Windows XP Professional SP3，CPU: Core2Duo E6750(2.66GHz)，メモリ: 2GB である．

上記の結果はクラフチック法を適用した後の結果である．しかし，上記の結果は区間行列の半径が大きく，そのままでは全探索を用いた候補集合の作成はできず，数値的最適 LQ 制御器を探すことができない．

次に，この問題を多倍長演算と精度保証付き数値計算を用いて解く．用いた多倍長の桁数は 10 進数で 115 桁である．求まったリカッチ方程式の解を以下に示す．ただし，スペースの都合から始めの 17 桁だけを示す．

```
===  IP中心  (   3  x   3)  NumericalMatrix  ===
                (  1)                    (  2)
(1)  4.6666666666667407e+12 1.3333333333334074e+12
(2)  1.3333333333334074e+12 4.0000000000000740e+12
(3) -3.7036925926036008e-2 -1.3333333333333703e+12
                (  3)
(1) -3.7036925926036008e-2
(2) -1.3333333333333703e+12
(3)  3.3333333333333518e+12
===  IP半径 ( 3 x 3) NumericalMatrix ===
                (  1)                    (  2)
(1) 6.2507122960749413e-102 5.8321378119627800e-102
(2) 6.3902371241123283e-102 6.3065222272898961e-102
(3) 5.5492674927977715e-102 5.2740384998132317e-102
                (  3)
(1) 5.4813333361817674e-102
(2) 5.7205179495328703e-102
(3) 5.1345136717758446e-102
```

区間行列の半径が10進数で約10^{-100}という値で求まっている．このように多倍長演算を用いることで，通常の倍精度演算では不可能な高精度の解を求めることができる．

リカッチ方程式の解を精度保証付き多倍長計算で求めるのにかかった時間は，0.984秒である．ただし，計算機環境は前記の環境と同じである．

つぎに，この問題について数値的最適LQ制御問題を解く．

(S1) 上記の結果と同じ精度で多倍長区間行列を求め，倍精度区間行列に近似すると，

```
===  IP中心  (   3  x   3)  Matrix  ===
                (  1)                    (  2)
(1)  4.6666666666667410e+12  1.3333333333334075e+12
(2)  1.3333333333334075e+12  4.0000000000000740e+12
(3) -3.7036925926036000e-02 -1.3333333333333704e+12
                (  3)
(1) -3.7036925926036000e-02
(2) -1.3333333333333704e+12
(3)  3.3333333333333520e+12
===  IP半径 ( 3 x 3) Matrix ===
                (  1)                    (  2)
(1) 9.7656250000000000e-04  2.4414062500000000e-04
(2) 2.4414062500000000e-04  9.7656250000000000e-04
(3) 6.9388939039072280e-18  2.4414062500000000e-04
              ( 3)
(1) 6.9388939039072280e-18
(2) 2.4414062500000000e-04
(3) 4.8828125000000000e-04
```

となる．求まった区間行列IPは，$(2,2)$成分のみ区間内に3個の浮動小数点数があり，その他の区間には2個の浮動小数点がある．リカッチ方程式の解は対象行

列であるので，候補集合の要素は $2^5 \times 3 = 96$ 個である．
(S2) 96 個の候補の内，妥当でないとはいえないリカッチ方程式の解の個数は 36 個であった．
(S3) これらのリカッチ方程式の解について (4.6) 式を計算する．(4.6) 式を最も小さくする解の各成分は

$$\begin{aligned} p_{11} &= 4.6666666666667410 \times 10^{12} \\ p_{12} = p_{21} &= 1.3333333333334072 \times 10^{12} \\ p_{13} = p_{31} &= -3.7036925926036000 \times 10^{-2} \\ p_{22} &= 4.0000000000000737 \times 10^{12} \\ p_{23} = p_{32} &= -1.3333333333333704 \times 10^{12} \\ p_{33} &= 3.3333333333333520 \times 10^{12} \end{aligned}$$

であった．そのときの (4.6) 式の値は，

$$5.855351574598118543 46 \times 10^{-3}$$

である．また，上記の値を (4.4) 式の左辺に代入し残差ノルムを求めると，

$$2.778390817270429948 08 \times 10^{3}$$

となり，通常の倍精度のみを用いた結果の中心の値よりも，小さい値が求まっている．

このように多倍長演算を用いることで，通常の倍精度による演算では不可能であった数値的最適 LQ 制御器を求めることが可能である．なお，多倍長演算のみを用いた結果を (4.4) 式の左辺に代入し残差ノルムを求めると，

$$3.515938942373840943 69 \times 10^{-94}$$

となった．

今回の問題を解くのにかかった時間を表 5.6 に示す．計算機環境は，前記の環境と同じである．

表 5.6: Computational Time

Step	Time(ms)
Step1(Computation of interval P + select candidates of optimal P)	1016.0 (1000.0+16.0)
Step2	11359.0
Step3	1094.0
Total	13469.0

第6章　提案手法の実装

6.1　精度保証付き数値計算パッケージ

6.1.1　JCGA

本研究では Java 精度保証付き数値計算パッケージとして JCGA(Java Computing Guaranteed Accuracy) を開発した．JCGA は，区間演算，区間行列演算 [4–6, 29, 39]，線形方程式 [4–6, 30, 39]，非線形方程式 [4–6, 39]，固有値問題 [4, 32]，多項式の評価 [30]，関数の微分 [4, 5, 30] の精度保証付き数値計算を行うことができる。

JCGA では複素数演算や行列演算などの基本演算に Java 数値計算パッケージ NFC[37] を，多倍長演算に MPFloat[20] を利用している．図 6.1 に精度保証付き数値計算と多倍長演算を組み合わせた数値計算環境の構成を示す．

6.1.2　CPU の丸めモード変更による高速区間作成

ここでは浮動小数点数に基づく精度保証付き数値計算で用いられる，CPU の丸めモード変更を用いた高速区間作成について説明する [5].

われわれがコンピュータで計算を行う場合，従来は最近点への丸め (実数 r に最も近い浮動小数点数に丸める) という CPU の丸めモードが用いられる．このほかにも丸めモードには，下向きの丸め (実数 r 以下の浮動小数点数の中で最も大きい数に丸める)，上向きの丸め (実数 r 以上の浮動小数点数の中で最も小さい数に丸める) などがある [35].

そこで浮動少数点数に基づく精度保証付き数値計算では，図 6.2 に示すように CPU の丸めモードを制御し，浮動小数点数を両端とし，実数の真の値を包み込む区間を作成する．

この CPU モードの変更は通常，加算または乗算と同じ計算コストで実行できる．そのため，区間の下限と上限それぞれを求める前に丸めモードを変更し，その値を計算すれば，丸め変更にかかるコストは無視できる．これにより，高速に区間を作成することができる．

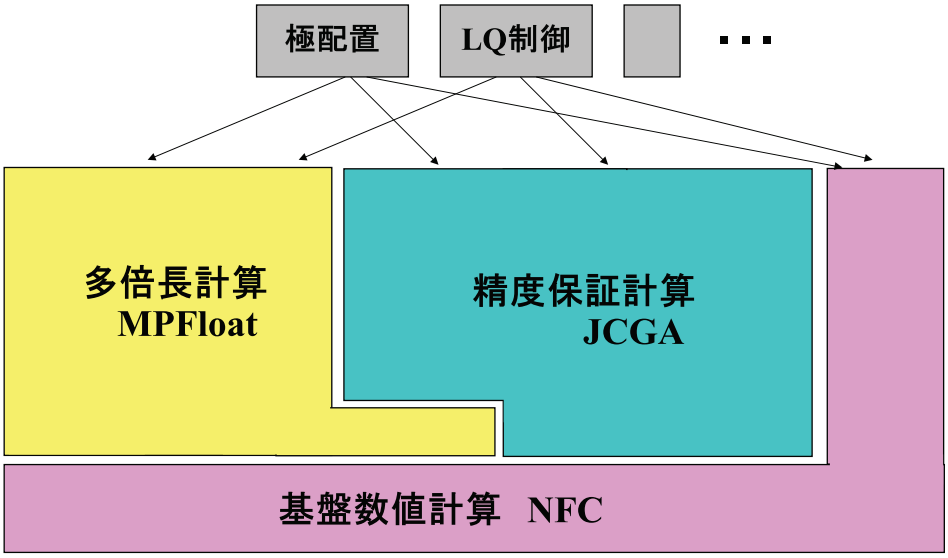

図 6.1: Architecture of numerical computation environment

Java は CPU の丸めモードを変更する方法を提供していない [40] ので，JCGA は JNI[41] で C 言語のコード [5] を呼び出すことで丸めモードの制御を行っている．

6.1.3　JCGA のアーキテクチャ

図 6.3 に JCGA のアーキテクチャ（パッケージ図）を示す．interval パッケージは区間演算，区間行列演算などを行う．interval パッケージは CPU の丸めモード制御を担当する round パッケージ，内部の基本演算を行う nfc パッケージを利用する．

nfc パッケージ内では一般的な数値を表すインターフェース NumericalScalar が定義されている．そして，このインターフェースを実装し倍精度を表す DoubleNumber クラスが実装されている．また，mpfloat パッケージでは NumericalScalar インターフェースを実装し，多倍長精度を表す MPFloat クラスが実装されている．これにより，数値を扱う際に倍精度か多倍長精度か具体的に記述せず，NumericalScalar という抽象的なレベルでコードを書くことができる．cga パッケージでは，この仕組みを用いて倍精度と多倍長精度を容易に切り替えることができる．そして，具体的な計算を行う際の初期値として double 型（DoubleNumber クラス）を用いると倍精度で，MPFloat クラスを用いると多倍長精度で計算が行われる．

また，JCGA には線形方程式，非線形方程式，固有値問題，多項式の評価，関数の微分の精度保証付き数値計算を行うためのそれぞれ linear，nonlinear，eigen，

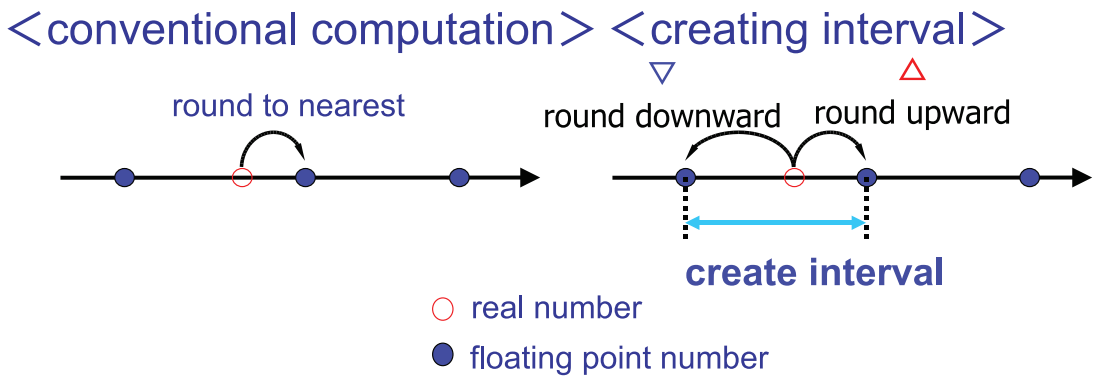

図 6.2: Conventional computation and verified numerical computation

表 6.1: Classes of numerical computation with guaranteed accuracy

精度保証計算	パッケージ	主要クラス	アルゴリズム
区間演算	interval	Interval	2.2 節
区間行列演算	interval	IntervalMatrix	2.2 節
関数の微分	derivative	IntervalDerivative	2.3.1 節
線形方程式	linear	LinearEquationVerifier	2.1 節
非線形方程式	nonlinear	NonLinearEquationVerifier	2.4.1 節
固有値問題	eigen	EigenVerifier	2.4.2 節
多項式の評価	polynomial	PolynomialVerifier	2.4.3 節

polynomial，derivative パッケージがあり，これらのパッケージは interval パッケージを利用する．そして，これらのパッケージは主要クラスとして，表 6.1 に示す精度保証計算を行うためのクラスを含んでいる．

6.1.4 JCGA を用いた計算例

ここでは JCGA を用いた精度保証付き数値計算の例として，固有値の精度保証付き数値計算を示す．文献 [32] のアルゴリズムが実装されている EigenVerifier クラスを用いて行列

$$A = \begin{bmatrix} -4 & 17 & 60 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

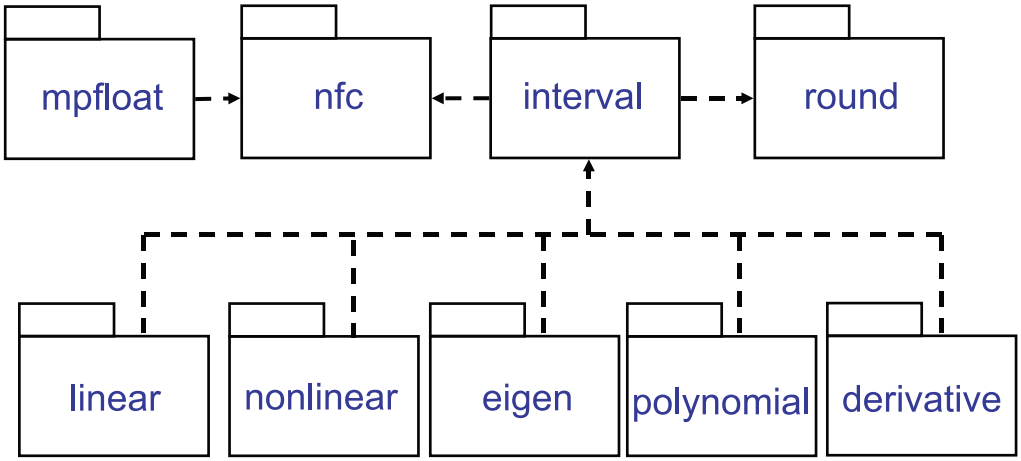

図 6.3: Architecture of JCGA

の固有値の精度保証付き数値計算を行う．この行列の真の固有値は，$\{4, -3, -5\}$ である．以下にプログラムと計算結果を示す．

固有値の精度保証プログラム

```
public class EigenVerifierSample {
  public static void main(String[] args) {
    //行列 A
    Matrix A = new DoubleMatrix(new double[][] {
               {-4, 17, 60},
               { 1,  0,  0},
               { 0,  1,  0}});
    //精度保証付き固有値を計算
    EigenVerifier verifier = new EigenVerifier();
    IntervalMatrix ans = verifier.solve(A)[0];
    //中心と半径の表示
    ans.printMidRad("円盤（固有値の存在範囲）");
  }
}
```

精度保証付き固有値

```
===  円盤（固有値の存在範囲）中心  (   3  x   1)  CoMatrix  ===
        [ (  1)-Real                  (  1)-Imag ]
 (  1)  4.000000000000000000E+00  0.000000000000000000E+00
 (  2) -3.000000000000000400E+00  0.000000000000000000E+00
 (  3) -5.000000000000000000E+00  0.000000000000000000E+00
===  円盤（固有値の存在範囲）半径  (   3  x   1)  Matrix  ===
            (  1)
 (  1)  2.329383478607396300E-16
 (  2)  1.437330003186217900E-15
 (  3)  1.258978188088645200E-15
```

計算結果は複素数の存在範囲を円盤で表しており，上記の中心と半径を持つ円盤の上に真の解が存在することを示す．半径が倍精度演算の限界に近い値で求まっており，精度の良い解が求まっていることがわかる．

表 6.1 中の他のクラスも同様に簡単に利用できる．また，多倍長演算を行う際は，行列の初期化を行う際に MPFloat 型の初期値を与えるだけで多倍長計算を行うコードに簡単に変更できる．表 6.1 中のクラスは組み合わせて利用できるので，多くの問題の解を多倍長精度保証付きで求めることができる．

6.1.5 精度保証付き LQ 制御問題の解法の実装

JCGA を用いて LQ 制御問題を精度保証付きで解く LQRVerifier クラスと数値的最適 LQ 制御問題を解く LQRVerifiedOptimizer クラスを作成した．LQRVerifier クラスは IntervalMatrix，IntervalDerivative，LinearVerifier，NonLinearVerifier，EigenVerifier を用いる．LQRVerifiedOptimizer クラスは Interval，IntervalMatrix，EigenVerifier，LQRVerifier を用いて記述されている．

LQRVerifier クラスを用いて

```
LQRVerifier verifier = new LQRVerifier(A,B,Q,R);
verifier.solve();
IntervalMatrix F = verifier.getF();
IntervalMatrix P = verifier.getP();
```

のように記述することで，LQ 制御問題を精度保証付きで解くことができる．

LQRVerifiedOptimizer クラスを用いて

```
VerifiedOptimalproblem problem
  = new LQRVerifiedOptimalProblem(A, B, Q, R);
VerifiedOptimizer optimizer
  = new LQRVerifiedOptimizer(problem);
CadidateSet cadidateSet
  = new FullSearchCandidateSet();
optimizer.setCadidateSet(cadidateSet);
optimizer.solve();
Matrix F = optimizer.getF();
Matrix P = optimizer.getP();
```

のように記述することで，数値的最適 LQ 制御問題を解くことができる．LQRVerifiedOptimalProblem クラスには問題と評価関数が記述されている．CandidateSet クラスは候補集合を作成するクラスであり，その子クラスである FullSearchCandidateSet クラスは全探索用の候補集合を作成する．

6.2　シミュレーションとリアルタイム制御実験の統合環境

数値的最適解を選択した後は，その制御器（数値解）を用いてシミュレーションと制御実験が行われる．シミュレーションや制御実験を行った結果，所望の結果が得られなかった場合は再び上流に戻り，モデリングや設計が行われる．そして，再モデリングや再設計が行われた後には，設計された検証付き制御器を用いて再びシミュレーションと制御実験が行われる．この繰り返し作業は，所望の結果が得られるまで繰り返される．このように，シミュレーションと制御実験は繰り返し行われる．本論文では，このシミュレーションと制御実験の繰り返し作業を効率的に行うための統合環境を開発した [42–46]．

シミュレーションとリアルタイム制御実験の統合環境は C 言語と Java 言語を用いて実装されている．図 6.4 に開発した統合環境の概要を示す．統合環境では，シミュレーションプログラムからフレームワークを用いた RT 制御プログラムが自動生成される．この統合環境ではユーザはシミュレーションとリアルタイム制御実験を統合して実行できるので，制御系設計プロセスの効率を向上させることができる．

図 6.5 に開発した統合環境のアーキテクチャを示す．RT 制御プログラム作成フレームワークは RTOS と Java VM 上で動作する．フレームワークには，RTOS の機能を使用するホットスポット①と，Java VM の機能を使用するホットスポット②がある．そして，フレームワーク上で動作する GUI を用いてシミュレーション，制御実験を行うことができる．また，シミュレーションと制御実験には自動生成

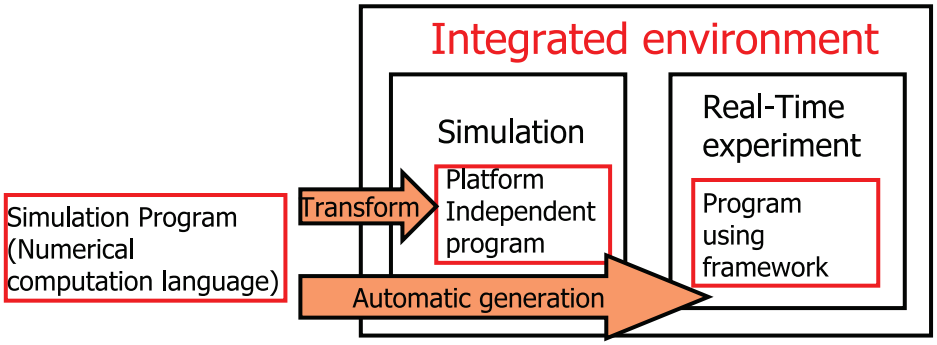

図 6.4: Schematic of Integrated Environment

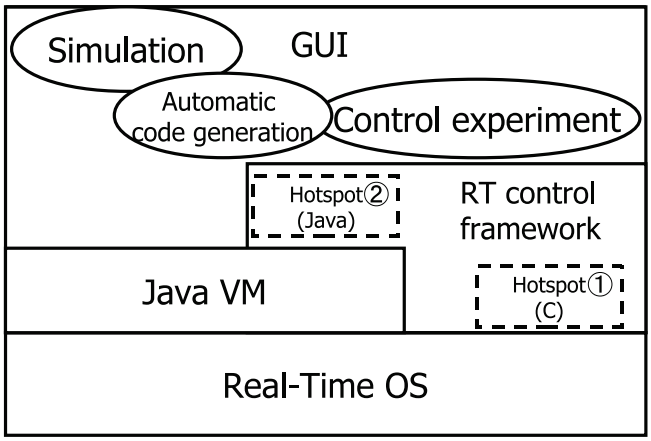

図 6.5: Architectrue of Integrated Environment

が利用される.

6.2.1 RT 制御プログラム作成フレームワーク

本節では RT 制御プログラム作成の枠組みを提供する RT 制御プログラム作成フレームワークを提案する.

本論文で提案するフレームワークのアーキテクチャを図 6.6 に示す. RT 制御プログラム作成フレームワークは, RT 制御プログラム作成の枠組みを提供する. RT 制御プログラム作成フレームワークを用いることで, 制御対象に依存するコードが明白になり, 集中管理できる. また, 制御対象に固有の部分のみ作成すれば, RT 制御プログラムを作成できる. これにより RT 制御プログラムの作成が容易になり, 作業効率を上げることができる. 制御対象に固有の部分のうち, リアルタイ

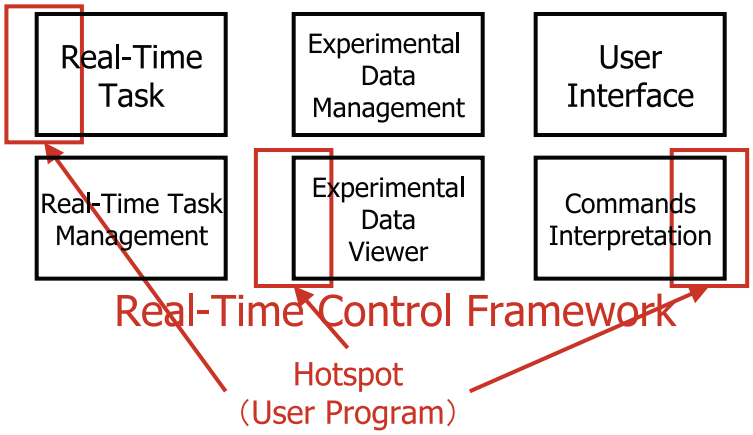

図 6.6: Architecture of RT control program framework

ムタスク，データの表示，ユーザから送られてきたデータの解釈の一部分を作成するだけで，RT 制御プログラムを作成できる．アーキテクチャの詳細は以下のようになっている．

- リアルタイムタスクモジュール
  リアルタイム処理を行い，制御則の計算等を行う
- リアルタイムタスク管理モジュール
  リアルタイムタスク生成・管理を行う
- 実験データ取得モジュール
  実験データを管理する
- 実験データ表示モジュール
  実験データを表示する
- ユーザインターフェースモジュール
  制御の開始・終了等のユーザからのコマンドを受け付ける
- コマンド解釈モジュール
  ユーザから送られてきたコマンドを解釈し，それを処理する

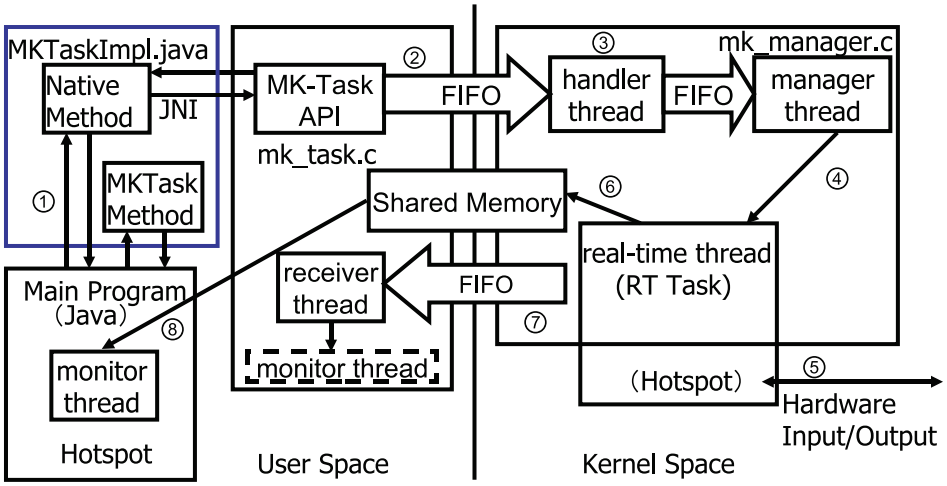

図 6.7: Architecture of MK-Task

ReTiCoF

本論文で提案する RT 制御プログラム作成フレームワークの実装として，ReTiCoF（Real Time Control Framework）を開発した [42]．ReTiCoF はリアルタイム OS の一つである RT-Linux[47] 上で動作し，C 言語と Java 言語を用いて記述されている．ReTiCoF において実際にリアルタイム処理を行う部分（MK-Task[48]）のアーキテクチャを図 6.7 に示す。MK-Task は，C 言語によって実装されており RT-Linux の機能を簡単に利用できる仕組みを提供する．

MK-Task はユーザ空間で動作する mk_task.c と，カーネル空間で動作する mk_manager.c から成る．mk_task.c は MK-Task の API 群とカーネル空間から送られてくるデータを受け取る receiver スレッドのコードを含む．mk_manager.c はユーザ空間からの要求があると起動する handler スレッド，handler スレッドが受けた要求通りにリアルタイムタスクを管理する manager スレッド，さらに実時間制約を持つスレッドとして内部に RT タスクを持つ．また MKTaskImpl.java は JNI 経由で MK-Task の API 群を呼び出すネイティブメソッドと，Java から MK-Task を利用する際に必要なメソッドを持つクラスである。JNI（Java Native Interface）[41] は，Java から C 言語などのネイティブコードを利用するための技術である．

MK-Task を用いてリアルタイム制御を行う場合の，処理の流れを次に示す．

1. Java のメインプログラムから MK-Task の関数を MKTaskImpl クラスの native メソッドとして呼び出す．

2. MK-Task の関数を用いたユーザ空間からの要求が，RT-FIFO 経由でカーネ

ル空間に送られる．

3. mk_manager.c 内の handler スレッドが起動する．
4. handler スレッドが受け取った要求が manager スレッドに渡され，manager スレッドが RT タスクを要求通りに管理する．
5. RT タスクがハードウェア入出力を行い，リアルタイム制御を行う．
6. リアルタイム制御の実験データがユーザ空間とカーネル空間の共有メモリに書き込まれる．
7. receiver スレッドが RT タスクから RT-FIFO 経由でデータの総数を受け取る．
8. monitor スレッドが共有メモリからデータを受け取り，データの表示を行う．

6.2.2　オブジェクトモデルを用いたプログラムの変換

本節ではオブジェクトモデルとデザインパターン [49] を用いたシミュレーションプログラムの変換を提案する．まず，数値計算言語で記述されたシミュレーションプログラムをプラットフォームに依存しない言語で記述されたシミュレーションコードに変換する手法を提案する。また，Factory Method パターン [49] を用いてリアルタイム処理コード（RT 制御プログラム作成フレームワークのホットスポット）を生成する手法を提案する．数値計算言語で記述されたシミュレーションプログラムから，RT 制御プログラムを自動生成できるので，制御系設計プロセスが効率化される．RT 制御プログラム作成フレームワークのホットスポットを生成することで，生成するプログラムを制御対象に固有の部分のみに限定できる．また，シミュレーションプログラムをプラットフォームに依存しない言語に変換することで，シミュレーションと制御実験を同じプラットフォーム上で実行できるようになる．

図 6.8 にオブジェクトモデルを用いたシミュレーションプログラムの変換の概要を示す．まず，Parser が数値計算言語で記述されたソースコードを受け取る．Parser は構文解析を行い，文法規則に基いて数値計算言語のソースコードの情報を持ったオブジェクトツリーを構築する．ツリーの構築には，プラットフォームに依存しないコードを生成する機能を定義したオブジェクトを利用する．そして，生成されたオブジェクトツリーからプラットフォームに依存しない言語のシミュレーションプログラムが生成される．

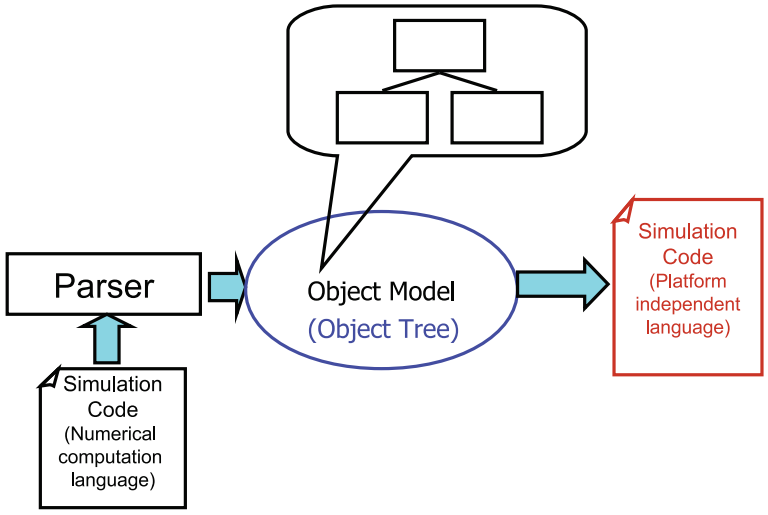

図 6.8: Transformation of simulation program using Object Model

また，オブジェクトツリーを構築する際に，Factory Method パターン [49] を用いれば，ツリーに組み込むオブジェクトを目的に応じて切り替えることが可能となる．

図 6.9 は Factory Method パターンを用いたシミュレーションプログラムの変換の概要を示しており，Parser がオブジェクトモデルを構築する部分に Factory Method パターンが適用されている．そして，オブジェクトモデル（Product）を Creator が作ると定めている．シミュレーションコードを生成する場合に使用されるオブジェクト群（ConcreteProduct）は Creator の実装クラスである SimCreator（ConcreteCreator）によって作成される．

図 6.10 は Factory Method パターンを用いた RT プログラムの自動生成の概要を示している．RT プログラムを自動生成するオブジェクト群（ConcreteProduct）は SimCreator のサブクラスである RTCreator（ConcreteCreator）によって作成される．このように，同じ構造のオブジェクトモデルを用いてプラットフォームに依存しないシミュレーションプログラムへの変換と，リアルタイム処理コードの自動生成が行える．また，Creator のサブクラスとオブジェクトツリーに組み込まれるクラス群を作成すれば，生成するコードの種類を追加できる．

今回開発した統合環境では数値計算言語として MATX[50] を，プラットフォームに依存しない言語として Java を採用した．統合環境は Java で実装された MATX イ

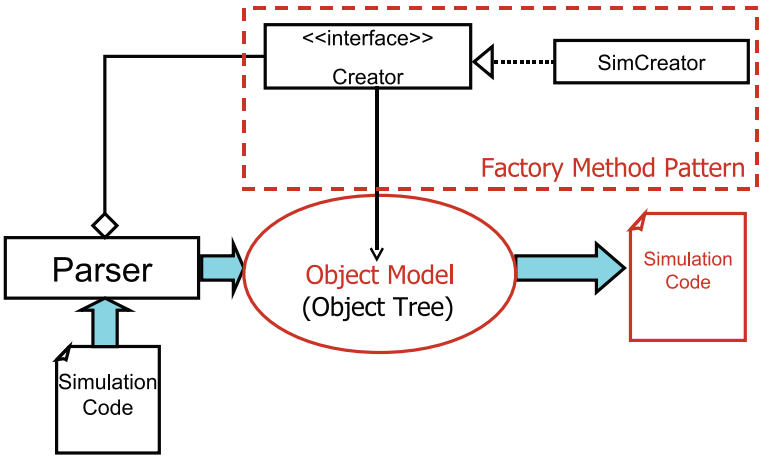

図 6.9: Transformation of simulation program using Factory Method Pattern

ンタプリタである matj[51] を用いて MATX のソースコードから MATX のオブジェクトモデルを作成し，オブジェクトモデルを順番に辿ることで，Java ソースコードを生成する．

また，matj の Parser に Factory Method パターンを適用し，MATX で記述されたシミュレーションプログラムからリアルタイム処理コード（RT 制御プログラム作成フレームワークのホットスポット部分）を自動生成できるようにした．具体的には，リアルタイム処理コード（行列計算，三角関数等）を生成するクラス群と，そのクラス群をオブジェクトツリーに組み込むクラスを作成した．

6.2.3　プラットフォーム依存部分の分離

本節では，RT 制御プログラムをプラットフォームに依存する部分と依存しない部分に分離する方法を提案する．そして，RT 制御プログラム作成フレームワークのプラットフォーム依存部分を分離する．

RT 制御プログラムのプラットフォーム依存部分の分離

一般に，RT 制御プログラムはリアルタイム処理が必要な部分と，リアルタイム処理が不要な部分から構成される．リアルタイム処理が必要な部分をプラットフォームに対応した言語で記述し，リアルタイム処理が不要な部分を，プラットフォームに依存しない言語で記述することで，リアルタイム処理が不要な部分を

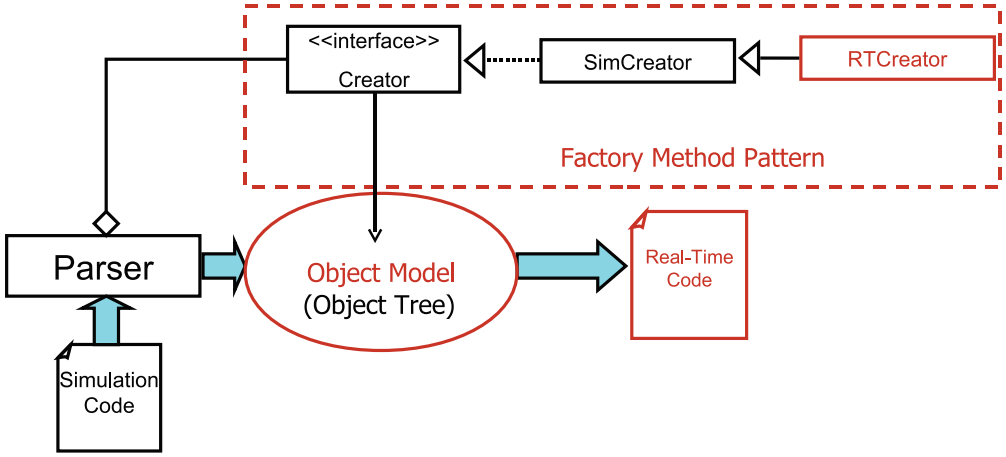

図 6.10: Automatic generation of Real-Time contorl program using Factory Method Pattern

プラットフォームから独立させることができる．RT 制御プログラム作成フレームワークのプラットフォーム依存部分の分離に関する概要を図 6.11 に示す．

サーバ・クライアントシステム

前節で述べた手法とネットワークを利用するサーバ・クライアント方式を組み合わせることで，得られるプログラムの移植性の向上について述べる．

図 6.12 にサーバ・クライアントシステムの概要を示す．制御対象はサーバマシンに接続されている．サーバマシンには RTOS がインストールされており，RT プログラムの実行が可能である．サーバマシンはネットワークを介してクライアントマシンと接続されている．

このシステムではリアルタイム処理はサーバマシンのみで実行され，クライアントマシンでは非リアルタイム処理のみが実行される．また，サーバマシンとクライアントマシンの間では，リアルタイム性を必要としない情報がやりとりされる．このため，クライアントマシンにはリアルタイムプログラムを実行するための特別な環境は必要ない．また，非リアルタイム処理部分はプラットフォーム非依存なので，クライアントマシンは特定の OS に依存しない．このようにプラットフォーム依存部分の分離とサーバ・クライアント方式を組み合わせることで，プログラムの移植性が向上する．

また，RT 制御プログラムがクライアントからサーバへ自動的に送信される仕組みを実現することで，RT プログラムの配備の問題を解決できる．

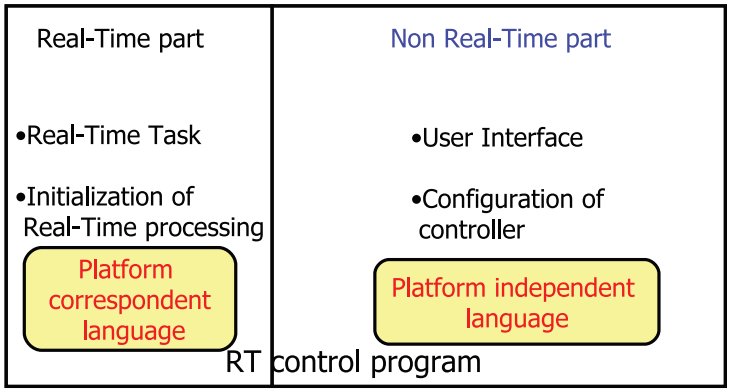

図 6.11: Separation of platform dependent parts

本研究ではプラットフォームに対応した言語として C 言語を，プラットフォームに依存しない言語として Java 言語を用いた．C 言語と Java 言語の併用は JNI を用いて実現した．

本統合環境では，Java で分散オブジェクトを実現する技術である RMI（Remote Method Invocation）[52] を用いて，サーバ・クライアントシステムを実現した．分散オブジェクトとは，ネットワークを介したメソッド呼び出しが可能なオブジェクトである．また，RT 制御プログラムをクライアントマシンからサーバマシンへ自動的に送信する仕組みを，RMI を用いて実現した．

6.2.4　ローカル・リモート兼用 GUI

図 6.13 に統合環境で提供される GUI を示す．シミュレーション・自動生成・制御実験に関するメニューやツールバー，実験結果・リアルタイムタスクのプロパティ・ユーザへのメッセージを表示する部分がある．

本統合環境では，1 台のマシンのみを用いるローカル制御と，サーバマシンとクライアントマシンを用いる遠隔制御で同一の GUI を使うことができる．1 個の GUI をローカル制御とリモート制御で兼用できるように，Proxy パターン [49] を用いている．Proxy パターンとは，実際の主体である RealSubject クラスの処理を Proxy クラスが代理人として処理するデザインパターンである．

図 6.14 に Proxy パターンを用いた統合環境のクラス図を示す．図 6.13 に示した GUI は RTWindow クラスによって提供される．RTWindow クラスは，ViewPanel クラス等の GUI の部品を提供するクラスと，リアルタイム制御を行う MKTask インターフェイスを実装したクラスを持っている．ローカル制御は MKTaskImpl クラ

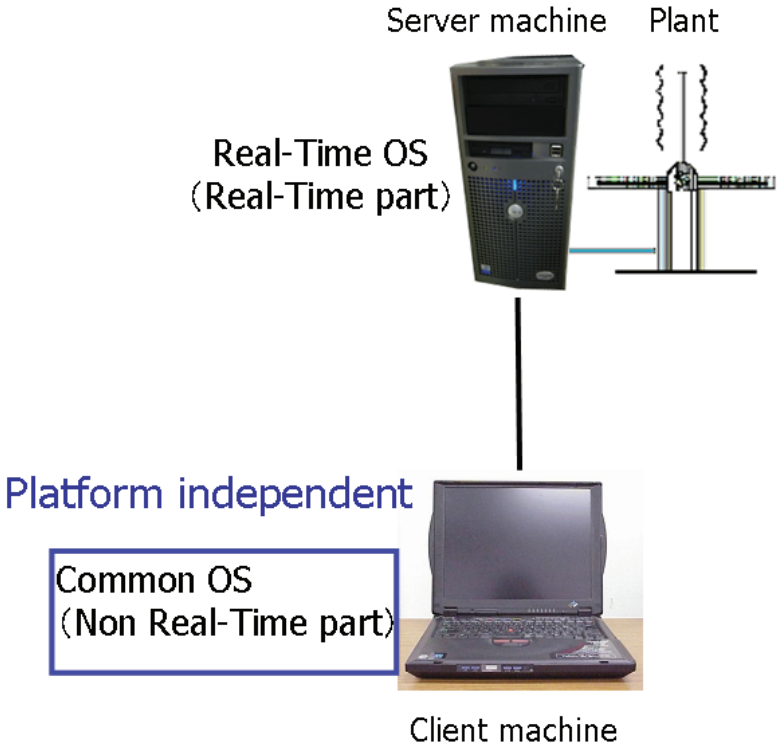

図 6.12: Server and client system

ス，リモート制御は分散オブジェクトである RemoteMKTaskImpl（RealSubject）の代りに RemoteMKTaskProxy（Proxy）クラスが使用される．

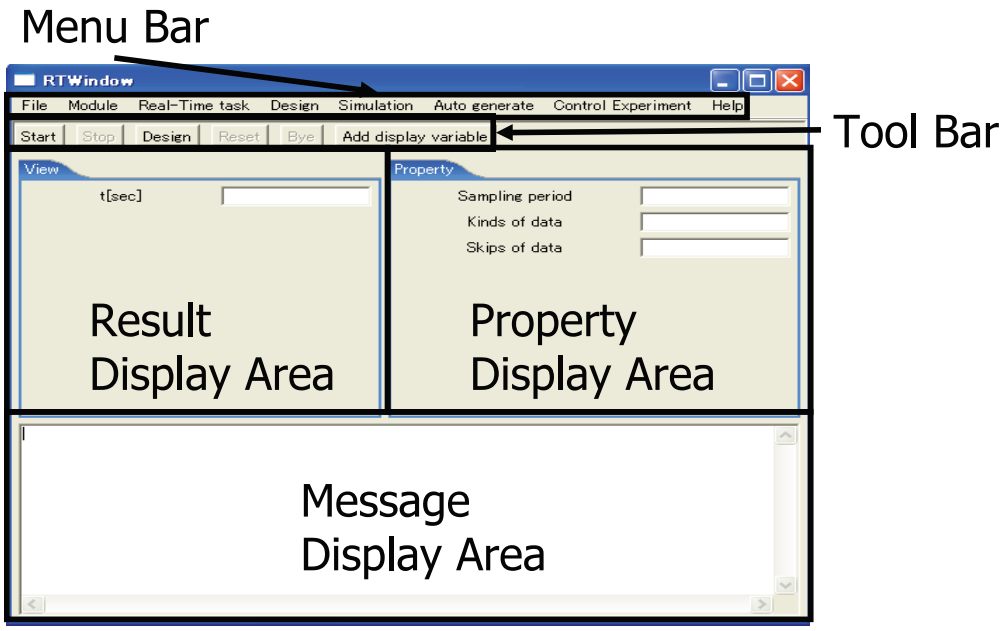

図 6.13: GUI of integrated environment

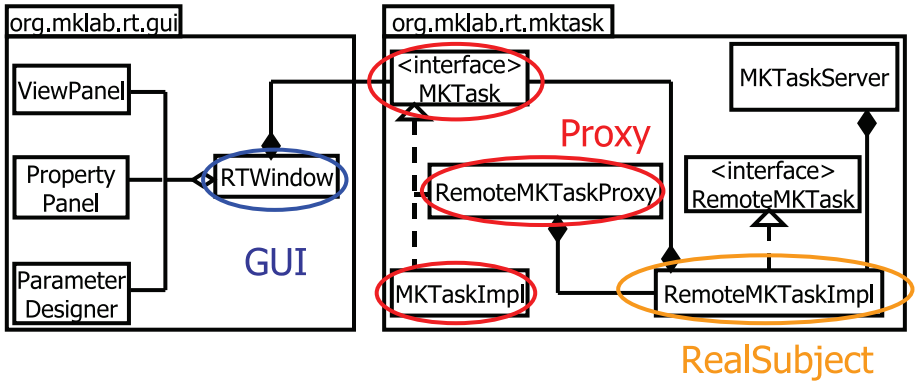

図 6.14: Class Diagram of Real-Time control system

第7章　まとめ

本研究では，精度保証付き数値計算に基づく検証付き制御系設計を行う手法を提案した．特に状態フィードバック制御問題，極配置問題と LQ 制御問題を検証付きで解く手法を提案した．今回提案した手法を用いることにより，制御系設計プロセスにおける制御系解析，及び制御器設計における数値計算誤差の問題の多くを解決し，設計仕様を満たすことの検証を行った検証付き制御系を設計することが可能となった．今後は実際の制御対象への適用や，精度指定付き数値計算 [53] の適用，他の設計問題への適用，計算の高速化及び記述の容易化が必要である．

設計の前に行われる制御系解析を精度保証付きで行う手法を提案し，多倍長演算と組み合わせることで制御系の解析を高品質保証付きで行えるようになった．これまで数値計算誤差の問題により，不安定や不可制御であるはずのシステムが安定や可制御と判定されてしまう問題を解決した．また，本提案手法を用いることによって，正定性を用いた繰り返し計算の終了判断や多倍長計算をいつ用いるかの判断を厳密に行うことが可能になった．行列の正定性の判定は，制御器の検証にも用いられる．

状態フィードバック制御問題を検証付きで解く手法として，極配置問題と LQ 制御問題を検証付きで解く方法を提案した．提案手法により，検証付き制御器を設計できるようになった．また，検証付きの制御器の集合から，最も設計仕様を満たす制御器を選択する手法である数値的最適制御問題とその解法を提案した．

数値的最適制御問題を高速に解く手法として，GA を用いる手法と多倍長演算を用いる手法を提案した．多倍長演算に基づく精度保証付き数値計算を用いることで，通常の倍精度演算では不可能な高精度保証付きの解を求めることができる．そして，求まった多倍長の区間行列を倍精度に丸めることで，数値的最適制御器の候補集合の要素を大幅に減らすことができる．

また，精度保証付き数値計算が容易に行えるように Java 精度保証付き数値計算パッケージを開発した．開発したパッケージと多倍長演算パッケージを組み合わせて使用することで，多くの問題を高精度かつ精度保証付きで解くことができる．

数値的最適制御器を選択した後に行われる，シミュレーションと制御実験の繰り返し実行作業を効率化する手法を提案した．具体的には，RT 制御プログラム作成フレームワーク，オブジェクトモデルを用いたプログラムの変換，プラットフォー

ム非依存部分の分離に基づくシミュレーションとリアルタイム制御実験の統合手法である．そして提案した手法を実装したシミュレーションとリアルタイム制御実験の統合環境を開発した．

今後は実際の制御対象に適用し，本論文で提案した手法の有効性を確認する必要がある．

また，本研究で提案した手法によって保証された精度は，事後保証された精度である．しかし，精度保証付き数値計算と多倍長演算を組み合わせた手法として，事前に指定された精度の結果を返す精度指定付き数値計算 [53] も提案されている．この手法を制御系設計へ適用することによって，所望の精度かつ精度保証付きの制御器を求めることが可能になる．今後は，この精度指定付き数値計算を制御系設計に応用していくことも考えている．

本論文では極配置問題と LQ 制御問題を検証付きで解く手法を提案したが，今後はロバスト制御をはじめとする他の設計問題に対して検証付き設計を行える手法を提案していきたいと考えている．特にロバスト制御問題を検証付きで解くことで，制御系設計の際に発生する誤差であるモデリング誤差と数値計算誤差の両方の問題を解決した，検証付き制御系を設計することが可能になる．今回提案したリカッチ方程式の精度保証付きアルゴリズムを用いることで，ロバスト制御問題の一つである $\mathcal{H}_\infty$ 制御問題も解くことが可能である．そこで，今後は $\mathcal{H}_\infty$ 制御問題をさらに効率的に精度良く解く手法や，$\mathcal{H}_\infty$ 制御器の検証条件と数値的に最適な制御器を選択するための評価関数を考える必要がある．また，多倍長演算を用いて SDP を高精度で解くことで，ロバスト制御問題を解く手法も考案されている [54]．そこで，SDP の精度保証を可能にすれば SDP に基づく制御系設計を高品質保証付きで行えるようになる．これも，今後の課題である．

実装の課題としては，計算の高速化が挙げられる．精度保証では区間の下限と上限を求める必要があるため，通常の近似計算より 2 倍以上の時間がかかる．この問題を解決するために，並列計算の手法を用いて求める区間の上限と下限を並列に計算し，高速化すること等が考えられる．また，精度保証付き数値計算の記述の容易化も行う必要がある．現在の実装では Java 言語を用いて精度保証付き数値計算を行う必要があるが，今後は数値計算言語に区間を表す文法を導入して拡張することや，精度保証付き数値計算言語を行うための専用言語の作成を考えている．

シミュレーションと制御実験の実装例では RTOS として RT-Linux を用いたが，Linux カーネル 2.6 上で RT 制御プログラムを動作させる手法 [55–57] が提案されているので，今後は Linux カーネル 2.6 に統合環境を対応させることを考えている．

謝辞

本研究において，研究のご指導だけでなく，博士後期課程三年間の生活を通してさまざまな面でご指導頂いた九州工業大学大学院情報工学研究院システム創成情報工学研究系古賀雅伸准教授に心より感謝致します.

本論文をまとめるにあたり，有益なご指摘，ご討論をいただきました九州工業大学大学院情報工学研究院システム創成情報工学研究系延山英沢教授，同研究院知能情報工学研究系瀬部昇教授，同研究院情報創成工学研究系吉田隆一教授に感謝の意を申し上げます.

精度保証付き数値計算に基づく制御系設計に関して熱心に議論して頂いた新潟大学工学部情報工学科管野政明准教授に感謝致します.

知的クラスタテーマ７の皆様に感謝致します.

いつも楽しい研究室にしてくれた，古賀研究室の皆さんに感謝します.

最後に，家族に感謝致します.

参考文献

[1] 劉康志. 線形ロバスト制御. コロナ社, 2001.

[2] 浅井徹. ロバスト制御について, 2007. SICE 九州フォーラム 2007『フレッシュマンのための制御工学』.

[3] Nicholas.J.Higham, Mihail Konstanitinov, Volker Mehramann, and Petko Petkov. The sensitivity of computational control problems. *IEEE Control Systems Magazine*, Vol. 24, No. 1, pp. 28–43, February 2004.

[4] 日本数学会. 218 精度保証付き数値計算法. 岩波数学辞典 第 4 版, pp. 642–647. 岩波書店, 2007.

[5] 大石進一. 精度保証付き数値計算. コロナ社, 2000.

[6] 中尾充宏, 山本野人. 精度保証付き数値計算 チュートリアル:応用数理最前線. 日本評論社, 1998.

[7] 中尾充宏. 計算の信頼性評価　現代技術への数学入門シリーズ. 講談社サイエンティフィック, 2008.

[8] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis*. Springer, 2001.

[9] Masaaki Kanno. *Guaranteed Accuracy Computations in Systems and Control*. PhD thesis, University of Cambridge, 2003.

[10] 管野政明, 原辰次. H2 最適追従制御設計問題の精度保証付き計算アルゴリズム. 計測自動制御学会論文集, Vol. 43, No. 2, pp. 102–109, 2007.

[11] 管野政明. SISO H ∞ループ整形問題の準最適制御器の精度保証付き計算. シミュレーション, Vol. 25, No. 3, pp. 185–193, 2006.

[12] 管野政明, 内山裕貴, 原辰次. 離散時間 H2 最適制御問題に対する精度保証付き計算. システム制御情報学会論文誌, Vol. 20, No. 11, pp. 451–453, 2007.

[13] 古賀雅伸, 廣木聡憲. 精度保証付き数値計算に基づく制御系設計, 2005. 第5回計測自動制御学会制御部門大会 pp.707-710.

[14] 矢野健太郎, 古賀雅伸. 精度保証付き数値計算に基づく LQ 制御問題, 2007. 第36回制御理論シンポジウム, pp.271-274.

[15] Kentaro Yano and Masanobu Koga. Pole Assignment Method Based on Numerical Computation with Guaranteed Accuracy, 2007. SICE Annual Conference 2007 International Conference on Instrumentation, Control and Information Technology, pp.2960-2965.

[16] 矢野健太郎, 古賀雅伸. クラフチック法を用いた高精度な精度保証付き数値計算に基づく LQ 制御問題, 2008. 第8回制御部門大会.

[17] Kentaro Yano and Masanobu Koga. LQ Control Problem Based on Numerical Computation with Guaranteed Accuracy, 2008. 17th IFAC World Congress, MoC1.1.

[18] 矢野健太郎, 古賀雅伸. LQ 制御問題の精度保証付き数値計算. 計測自動制御学会論文集, Vol. 45, No. 5, pp. 261–267, 5 2009.

[19] 藤原宏志, 磯祐介. 高速多倍長計算環境における数値解析. 日本応用数理学会論文誌, Vol. 15, No. 3, pp. 403–417, 2005.

[20] 山村英介, 古賀雅伸, 矢野健太郎, 山田賢治. 多倍長計算を用いた制御系設計パッケージ, 2008. 第52回システム制御情報学会研究発表講演会 (SCI'08), pp.139-140.

[21] 山本野人, 松田望. 多倍長演算を利用した Bessel 関数の精度保証付き数値計算. 日本応用数理学会論文誌, Vol. 15, No. 3, pp. 347–359, 2005.

[22] 白石昌武. 入門現代制御理論. 日刊工業新聞社, 2002.

[23] William L. Brogan. *Modern Control Theory Third Edition.* Prentice-Hall International, Inc., 1981.

[24] Petko Hr. Petkov, Mihail M. Konstantinov, Da-Wei Gu, and V. Mehrmann. Numerical Solution of Matrix Riccati Equations: A Comparison of Six Solvers, 1999. NICONET Report.

[25] 矢野健太郎, 古賀雅伸. 精度保証付き数値計算に基づく制御系解析, 2007. 第50回自動制御連合講演会, pp.894-895.

[26] Juraj Hromkovic. *Algorithmics for Hard Computing Problems:Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Springer, 2001.

[27] Kentaro Yano, Masanobu Koga, and Eisuke Yamamura. Control System Design Method Based on Multiple-Precision Arithmetic with Guaranteed Accuracy, 2008. SICE Annual Conference 2008 International Conference on Instrumentation, Control and Information Technology, pp.1772-1777.

[28] 矢野健太郎, 中島大雅, 古賀雅伸. 多倍長演算を用いた高精度な精度保証付き数値計算に基づく LQ 制御問題, 2009. 第 9 回制御部門大会.

[29] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[30] 大石進一. 応用解析セミナー 数値計算. 裳華房, 1999.

[31] 久保田光一, 伊理正夫. アルゴリズムの自動微分と応用. コロナ社, 1998.

[32] Siegfried M. Rump. Computational error bounds for multiple or nearly multiple eigenvalues. *Linear Algebra and its Applications*, Vol. 324, pp. 209–226, 2001.

[33] 吉田哲基. 多重および近接固有値をもつ楕円型固有値問題に対する精度保証付き数値計算, 2000.

[34] The MathWorks Matlab. http://www.mathworks.com/products/matlab/.

[35] ANSI/IEEE. IEEE Standard for Binary Floating Point Arithmetic, 1985. Std 754-1985 edition, New York.

[36] 疋田, 小山, 三浦. 極配置問題におけるフィードバックゲインの自由度と低ゲインの導出. 計測自動制御学会論文集, Vol. 2, No. 5, pp. 556–560, 1975.

[37] 古賀雅伸, 松木毅. OS 中立な数値計算ライブラリの開発と制御系設計への適用, 2003. 第 3 回計測自動制御学会制御部門大会.

[38] P. Benner, A. Laub, and V. Mehrmann. A Collection of Benchmark Examples for the Numerical Solution of Algebraic Riccati Equations I: Continious-Time Case, 1995. Tech. Report SPC 95 22, Fak. f. Mathematik, TU Chemnitz-Zwickau, 09107 Chemnitz, FRG.

[39] G.I.Hargreaves. *Interval Analysis in MATLAB*, 2002. Numerical Analysis Report No.416.

[40] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification, Second Edition*. Addison-Wesley Pub, 1999.

[41] Rob Gordon. JNI Java Native Interface プログラミング. ピアソン・エデュケーション, 1998.

[42] 古賀雅伸, 矢野健太郎. リアルタイム処理コードを自動生成する遠隔制御システム, 2004. FIT2004 第 3 回情報科学技術フォーラム,pp.433-434.

[43] 矢野健太郎, 古賀雅伸. プラットフォームに依存しないシミュレーションとリアルタイム制御実験の統合環境, 2006. 第 6 回計測自動制御学会制御部門大会 pp.423-426.

[44] 矢野健太郎, 古賀雅伸. フレームワークとオブジェクトモデルを用いた制御系シミュレーション・実験統合環境, 2006. FIT2006 第 5 回情報科学技術フォーラム,pp.417-418.

[45] Kentaro Yano and Masanobu Koga. Platform Independent Integrated Environment for Simulation and Real-Time Control Experiment, 2006. SICE-ICASE International Joint Conference 2006, pp.411-416.

[46] Kentaro Yano and Masanobu Koga. Integrated Environment of Simulation and Real-Time Control Experiment for Control System, 2007. ICM2007 IEEE International Conference on Mechatronics, ThA-1-A-3, pp.1-6.

[47] 舟木陸議, 羅正華. Linux リアルタイム計測/制御開発ガイドブック. 秀和システム, 1999.

[48] Masanobu Koga and Shinkichi Kawakami. Development of Portable Real-Time Remote Control System in Java and its Application to Swing Up Control of Inverted Pendulum, 2003. SICE Annual Conference 2003 in Fukui.

[49] 結城浩. Java 言語で学ぶデザインパターン入門. ソフトバンク, 2001.

[50] 古賀雅伸. 制御・数値解析のための MaTX. 東京電気大学出版, 2000.

[51] 松木毅. オブジェクトモデル化に基づく Java による数値計算エンジンの開発. Master's thesis, 九州工業大学大学院情報工学研究科, 2005.

[52] 植田龍男. Java のからくり 2-実践・強化編. 株式会社ＩＤＧジャパン, 2001.

[53] 中島大雅, 古賀雅伸, 矢野健太郎. 多倍長演算を用いた精度指定可能な制御系設計パッケージ, 2009. 第 53 回システム制御情報学会研究発表講演会 (SCI'09) pp.479-480.

[54] 古賀崇史. 多倍長計算を用いた半正定値計画問題に基づく制御系設計支援パッケージの開発. Master's thesis, 九州工業大学大学院情報工学研究科, 2009.

[55] 岸田和也, 古賀雅伸. Linux カーネル 2.6 を用いたリアルタイム制御パッケージの開発, 2005. 第 49 回システム制御情報学会研究発表講演会 pp.61-62.

[56] 谷口仁志, 古賀雅伸, 松本明紘, 田中俊行. Linux のリアルタイム制御パッケージを用いたブロック線図に基づくシステム制御学習環境, 2009. 第 53 回システム制御情報学会研究発表講演会 (SCI'09).

[57] Kentaro Yano, Masanobu Koga, and Hitoshi Taniguchi. USB-Bootable Learning Support System for Real-Time Control, 2009. 8th IFAC Symposium on Advances in Control Education ThB03,4.

付 録A　リファレンス

本研究で開発したJCGAのAPI仕様を示す．このリファレンスでは，主要なクラスの主要なコンストラクタ，フィールド，メソッドの説明を行う．

A.1　org.mklab.cga.roundパッケージ

FPUクラス

説明

CPUの丸めモードを変更するクラスです。

フィールド

public static final int ROUND_NEAR = 0 最近点への丸め
public static final int ROUND_DOWN = 1 下への丸め
public static final int ROUND_UP = 2 上への丸め

メソッド定義

public static native void setRoundMode(final int mode) CPUの丸めモードを変更します。
public static native int getRoundMode() 現在のCPUの丸めモードの整数値を取得します。0は最近点への丸め、1は下への丸め、2は上への丸めです。

A.2 org.mklab.cga.intervalパッケージ

Intervalクラス

説明

区間を表す抽象クラスです。実数の区間演算を行うRealIntervalクラスと複素数の区間演算を行うComplexIntervalクラスに継承されます。

メソッド定義

`public add(Interval interval)` 自身と区間の和を求めます。
`public add(double d)` 自身と倍精度浮動小数点数の和を求めます。
`public add(Scalar<?> c)` 自身とスカラの和を求めます。
`public subtract(Interval i)` 自身と区間の差を求めます。
`public subtract(double d)` 自身と浮動小数点数の差を求めます。
`public subtract(Scalar<?> c)` 自身とスカラの差を求めます。
`public subtract(Interval i)` 自身と区間の差を求めます。
`public multiply(double d)` 自身と浮動小数点数の積を求めます。
`public multiply(Scalar<?> c)` 自身とスカラの積を求めます。
`public multiply(Interval i)` 自身と区間の積を求めます。
`public divide(Interval i)` 自身と区間の商を求めます。
`public divide(double d)` 自身と浮動小数点数の商を求めます。
`public divide(Scalar<?> c)` 自身とスカラの商を求めます。

メソッド定義

`public NumericalScalar<?> getInfimum()` 下限を取得します。
`public NumericalScalar<?> getSupremum()` 上限を取得します。
`public NumericalScalar<?> getMiddle()` 中心を取得します。
`public NumericalScalar<?> getRadious()` 半径を取得します。
`public void printInterval()` 下限と上限を [下限, 上限] と表示します。
`public void printInfSup()` 下限と上限を表示します。
`public void printMidRad()` 半径と中心を表示します。

RealInterval クラス

説明

実数区間演算を行うクラスです。このクラスは、Interval クラスを継承しているため、基本的に Interval クラスと同じメソッドを持ちます。

コンストラクタ

`public RealInterval(NumericalScalar<?> x)` 区間の中心を設定します

`public RealInterval(NumericalScalar<?> inf,`
`NumericalScalar<?> sup)`
下限と上限を設定します。

フィールド

`private NumericalScalar<?> inf` 区間の下限
`private NumericalScalar<?> sup` 区間の上限
`privated NumericalScalar<?> mid` 区間の中心
`private NumericalScalar<?> rad` 区間の半径

ComplexInterval クラス

説明

複素数区間演算を行うクラスです。このクラスは、Interval クラスを継承しているため、基本的に Interval クラスと同じメソッドを持っています。

コンストラクタ

`public ComplexInterval(ComplexScalar<?> x)` 区間の中心を設定します

`public ComplexInterval(ComplexScalar<?> mid,`
`NumericalScalar<?> rad)`
中心と半径を設定します。

IntervalMatrix クラス

説明

区間行列を表す抽象クラスです。このクラスは、実数行列の区間演算を行う RealIntervalMatrix クラスと複素数行列の区間演算を行う ComplexIntervalMatrix クラスに継承されます。

フィールド

protected NumericalMatrixOperator<?> inf 区間の下限
protected NumericalMatrixOperator<?> sup 区間の上限
protected NumericalMatrixOperator<?> mid 区間の中心
protected NumericalMatrixOperator<?> rad 区間の半径

メソッド定義

public IntervalMatrix add(IntervalMatrix y) 自身と区間行列の和を求めます。
public IntervalMatrix add(Matrix y) 自身と行列の和を求めます。
public IntervalMatrix add(Interval y) 自身と区間の和を求めます。
public IntervalMatrix addElementWise(double d) 自身の要素と浮動小数点数の和を求めます。
public IntervalMatrix addElementWise(Scalar<?> c) 自身とスカラの和を求めます。
public IntervalMatrix subtract(IntervalMatrix c) 自身と区間行列の差を求めます。
public IntervalMatrix subtract(Matrix c) 自身と行列の差を求めます。
public IntervalMatrix subtract(Interval i) 自身と区間の差を求めます。
public IntervalMatrix subtractElementWise(double d) 自身の要素と浮動小数点数の差を求めます。
public IntervalMatrix subtractElementWise(Scalar<?> c) 自身の要素とスカラの差を求めます。

メソッド定義

`public IntervalMatrix multiply(IntervalMatrix i)` 自身と区間行列の積を求めます。
`public IntervalMatrix multiply(Matrix c)` 自身と行列の積を求めます。
`public IntervalMatrix multiplyElementWise(Interval i)` 自身の要素と区間の積を求めます。
`public IntervalMatrix multiplyElementWise(double d)` 自身の要素と浮動小数点数の積を求めます。
`public IntervalMatrix multiplyElementWise(Scalar<?> c)` 自身の要素とスカラの積を求めます。
`public IntervalMatrix divide(IntervalMatrix i)` 自身と区間行列の商を求めます。
`public IntervalMatrix divide(Matrix c)` 自身と行列の商を求めます。
`public IntervalMatrix divideElementWise(Interval i)` 自身の要素と区間の商を求めます。
`public IntervalMatrix divideElementWise(double d)` 自身の要素と浮動小数点数の商を求めます。
`public IntervalMatrix divideElementWise(Scalar<?> c)` 自身の要素とスカラの商を求めます。
`public NumericalMatrixOperator<?> getInfimum()` 下限を取得します。
`public NumericalMatrixOperator<?> getSupremum()` 上限を取得します。
`public NumericalMatrixOperator<?> getMiddle()` 中心を取得します。
`public NumericalMatrixOperator<?> getRadious()` 半径を取得します。
`public void printInterval()` 下限と上限を [下限, 上限] と表示します。
`public void printInfSup()` 下限と上限を表示します。
`public void printMidRad()` 半径と中心を表示します。

RealIntervalMatrix クラス

説明

実数行列区間演算を行うクラスです。このクラスは、IntervalMatrix クラスを継承しているため、基本的にはフィールド、メソッドともに IntervalMatrix クラスと同じです。

コンストラクタ

`public RealIntervalMatrix(Matrix x)` 区間の中心を設定します
`public RealInterval(Matrix inf, Matrix sup)` 下限と上限を設定します。

ComplexIntervalMatrix クラス

説明

複素数行列区間演算を行うクラスです。このクラスは、IntervalMatrix クラスを継承しているため、基本的にはフィールド、メソッドともに IntervalMatrix クラスと同じです。

コンストラクタ

`public ComplexIntervalMatrix(Matrix x)` 区間の中心を設定します
`public ComplexInterval(Matrix mid, Matrix rad)` 中心と半径を設定します。

A.3 org.mklab.cga.linear パッケージ

LinearEquationVerifier クラス

説明

線形方程式 Ax=b を精度保証付きで解くためのクラスです。

コンストラクタ

```
public LinearEquationVerifier()
```
引数なしコンストラクタ。

メソッド定義

```
public IntervalMatrix solve(IntervalMatrix A,
                            IntervalMatrix B)
```
線形方程式 $Ax = b$ の解 x を精度保証付きで求めます。A, B が区間行列の場合。

```
public IntervalMatrix  solve(Matrix A, Matrix B)
```
線形方程式 $Ax = b$ の解 x を精度保証付きで求めます。A, b が行列の場合。

```
public IntervalMatrix  solve(IntervalMatrix A, Matrix B)
```
線形方程式 $Ax = b$ の解 x を精度保証付きで求めます。A が区間行列で、B が行列の場合。

```
public IntervalMatrix  solve(Matrix A, IntervalMatrix B)
```
線形方程式 $Ax = b$ の解 x を精度保証付きで求めます。A が行列で、B が区間行列の場合。

A.4　org.mklab.cga.eigenパッケージ

EigenVerifier クラス

説明

固有値を精度保証付きで解くためのクラスです。

コンストラクタ

```
public EigenVerifier()
```
引数なしコンストラクタ。

メソッド定義

public IntervalMatrix solve(IntervalMatrix A) 区間行列 A の固有値を精度保証付きで求めます。
public IntervalMatrix solve(Matrix A) 行列 A の固有値を精度保証付きで求めます。

A.5 org.mklab.cga.polynomial パッケージ

PolynomialVerifier クラス

説明

多項式を精度保証付きで解くためのクラスです。

コンストラクタ

public PolynomialVerifier() 引数なしコンストラクタ。

メソッド定義

public IntervalMatrix solve(Matrix A, double x) 行列 A を係数としてもつ多項式に x を代入したときの値を精度保証付きで求めます。
public IntervalMatrix solve(double[] cofficient, double x) 配列 cofficient を係数としてもつ多項式に x を代入したときの値を精度保証付きで求めます。

A.6 org.mklab.cga.derivative パッケージ

IntervalDerivative クラス

説明

微分値を精度保証付きで求めるクラスです。自動微分法を実装しています。

コンストラクタ

`public IntervalDerivative(Interval x)` 自身が整数である場合、その値を設定します。
`public IntervalDerivative(Interval x, Interval dx)` 値とそのときの微分値を設定します。

フィールド

`private Interval x` 任意の区間
`private Interval dx` x の微分値の区間

メソッド定義

`public IntervalDerivative add(IntervalDerivaitive y)` 自身と自動微分型の値の和を求めます。
`public IntervalDerivative subtract(IntervalDerivaitive y)` 自身と自動微分型の値の差を求めます。
`public IntervalDerivative multiply(IntervalDerivaitive y)` 自身と自動微分型の値の積を求めます。
`public IntervalDerivative divide(IntervalDerivaitive y)` 自身と自動微分型の値の商を求めます。
`public Interval getX()` x を取得します。
`public Interval getDX()` x の微分値 dx を取得します。

IntervalDerivativeFunction インターフェース

説明

IntervalDerivative 型によって関数を表現するためのインターフェースです。

メソッド定義

```
IntervalDerivative[] func(IntervalDerivaitive y)
```
関数を表現します。

IntervalDerivativeFunctionEvaluation クラス

説明

IntervalDerivativeFunction インターフェースを実装して定義された関数に値を代入したときの計算値とその微分値を精度保証付きで求めるクラスです。

コンストラクタ

```
public IntervalDerivativeFunctionEvaluation
             (IntervalDerivativeFunction idf)
```
値を求めたい関数オブジェクトを設定します。

フィールド

```
private IntervalDerivaitiveFunction idfunc
```
関数定義オブジェクト

メソッド定義

```
public IntervalMatrix getValues(IntervalDerivaitive[] x)
```
関数に x を代入したときの値、そのときの微分値、ヤコビ行列を求めます。

A.7 org.mklab.cga.nonlinear パッケージ

NonLinearVerifier クラス

説明

非線形方程式の解を精度保証付きで求めるクラスです。

コンストラクタ

```
public NonLinearVerifier(IntervalDerivativeFunction idf)
```
関数オブジェクトを設定します。

フィールド

```
private IntervalDerivativeFunction idne
```
関数定義オブジェクト

メソッド定義

```
public IntervalMatrix solve(Matrix x)
```
初期値 x で非線形方程式の解を精度保証付きで求めます。