卒　業　論　文

二次元多重連結領域内における構造安定な非圧縮流れの木表現の可視化手法

2016SE024　亀谷 拓磨
2016SE076　田島 嘉人
2016SE090　渡辺 康平

指導教員　横山 哲郎

20yy 年 mm 月

南山大学 理工学部 ソフトウェア工学科

Title Foo and Bar

— Subtitle Bar or Foo —

2016SE024　KAMEGAI Takuma
2016SE076　TAJIMA Yoshihito
2016SE090　WATANABE Kouhei

Supervisor　YOKOYAMA Tetsuo

Month 20yy

Department of Software Engineering
Faculty of Science and Engineering
Nanzan University

要約

ここに本論文の要約を書く。要約は論文のエッセンスを抜き出したものであるので、ここで取り扱う問題、その問題を解決するための手法、および主な成果が書かれていなければならない。要約を読むだけで、論文の概要が分かり、読者にとって興味を抱く内容か否かが分かるようになっている必要がある。

日本語と英語の両言語で要約を書くが、必ずしも 1 対 1 に対応する文章になっている必要はない。それぞれにふさわしい表現があるからである。

Abstract

In this part, the abstract of this paper is described. Since ‘abstract’ means the essence or summary of the paper, the abstract should include the description on the problems treated in the paper, the author’s approach for solving those problems, and the main results. Readers will understand the outline of the paper, without reading other parts, and be able to decide whether they will have interests in the paper or not.

目次

第 1 章

はじめに

1.1　流体力学の背景

流体力学は，気体や液体の運動について取り扱う力学の主要な研究分野である．流体力学の歴史は古く，この学問が確立される以前から人々の生活に即したものとして発展してきた．近年では，コンピュータの発達により流体方程式から数値解を求め流れを精密に計算することが可能となった．この手法は数値解析と呼ばれ，事故や災害などの実験不可能な課題も計算できる利点を持った近年の主要な流体力学の解析手法である．

1.2　流体の離散解析

離散解析は流体解析の一手法であり，前節で述べた数値解析に対照的である．離散解析は流れをトポロジカルな観点から解析する．トポロジカルな観点に着目することは大域的な構造に着目するということを意味し，これにより流れの本質的な構造を抜き出すことが可能となる．つまり流れの大枠に着目した解析を行いたい場合，離散解析を用いることで効率的に流れの解析を進められるということである．この解析法の代表的なアプローチとして，流体をその構造安定性に着目して解析する方法がある．構造安定性は，力学系に小さな乱れが加わっても流れのトポロジーが変化しない性質のことである．トポロジーは位相幾何学の用語であり，この立場に立てば，連続的に変形できる図形は同じ形である．例えば，四角形と三角形は同じ形である．構造安定性は一般の流体にはあり得ないが，文献 [1] によれば，有限の制度や有限の時間における挙動に着目したり，本質的な構造に着目するなどの現実的な目的の範囲内であれば，現実の流体にもこの考え方が適応できる．構造安定性に着目すれば，構造安定性から流れの変化を考察することができるようになる．例えば，語表現の研究では流れのトポロジーに対して対応した文字列を定義することで，二つの流体構造の中間構造を文字列の変化として考察することが可能となった [2].

1.3　本研究の目的

本研究は，離散解析手法の一つである木表現に対して，図上への可視化手法を与える．ここで，木表現には種類がいくつか存在するが，本研究で扱う木表現は [3] によって与えられた手法である．また，以下で木表現とはこのアルゴリズムを指す．木表現は流線構造を代数的に扱い流れの解析を行うことができるが，その表現から直感的に二次元上の流れの形状を把握することは困難である．そのため，二次元の形状を得たい場合は解析者がその都度木表現を組み合わせて図化することになるが，木表現が複雑になればなるほどその変換も煩雑になり，ともすれば途中で間違った変換を行ってしまうこともあり得る．また，論文などに木表現の図を入れようとした場合，描画ソフトを用いて図を手作業で書く必要がある．本研究は木表現を図に自動的に変換する方法を与えることで，解析者の効率的で確実な木文法による流れの解析に寄与するものである．

1.4　アプローチ

本研究では，トポロジーを二次元上に自動的に表現するプログラムを作成するにあたり，最初に完成したプログラムが満たすべき条件を定義した．そして，その定義をもとに Python と Asynptote の 2 つのプログラミング言語を使用しプログラムを作成した．

1.5　役割分担

南山太郎は主にこの部分を担当し、第 1 章、第 2 章、および 3.1 節を執筆した。南山花子は主にあの部分を担当し、3.2 節、第 4 章、および付録 A と B の説明文を執筆した。

第 2 章

関連研究

2.1　語表現の研究

本研究で扱う木表現は，語表現の研究を発展させたものである．語表現の研究では，流線の位相的な構造を分類し，それぞれの流線構造を系統的に文字列で表現するアルゴリズムを与えた [2]．以下では，語表現とは [2] によって与えられた表現方法を指す．語表現を用いた例として，翼の揚抗比の時間変化を語表現によって表した研究がある [4]．語表現は，有界な多重連結領域上で非圧縮かつ非粘性で構造安定な流れを代数的に表すことができる．多重連結領域とは複数の障害物が含まれている領域のことである．ここでは非圧縮・非粘性という理想流体を仮定しているが，この仮定は現実の流体に対しての直接の適用に制限を与える．しかし，現実への応用に関しては文献 [4] でその適用方法が考えられている．流線構造を文字列で表現することで，流線構造を数学的に厳密に分類できるようになり流れの変動を特徴付けて捉えることができるようになる．また，代数的に扱うことができるため，流線の構造の特徴を説明するための共通言語として用いることもできる．文献 [2] によって与えられた語表現には同じ流線に複数の語表現を与えることができるという問題があった．そのため，文献 [5] によって自然な語表現を与えるアルゴリズムが与えられた．また，文献 [6] によって流れの向きを考慮した場合の語表現を与えるアルゴリズムが与えられた．

2.2　木表現の研究

木表現もその前提条件は語表現と共通であり，有界な多重連結領域上で非圧縮性かつ非粘性で構造安定である．木表現はその特性から，語表現より細かい流れの分類を可能とする．そのため流れの特徴を語表現より，多く捉えることを期待されている．実際，円盤状の非圧縮流の反転の解析を木表現によって行った研究では，語表現と比べた木表現の表現力の高さが確かめられた [7]．

2.3　その他の離散解析手法

語表現および木表現と関連のある離散解析手法として，コンレイ・モース分解，グラフクラスタリングがある [1]．これらの手法は，流体に対してそれぞれ異なるアプローチを持つが，流体をその構造安定性に着目して解析するという点では一致している．

2.4　流れの解析以外での離散解析の使われ方

2.5　一般的な描画ソフトにおいての描画方法

第 3 章

自動可視化への準備

3.1　可読性の定義

作成するプログラムには，木表現で作成されたトポロジーを確実に図に再現することが求められるが，さらに，再現されたトポロジーは可読性が高いものである必要もある．本研究では，可読性の高さを定義する上でグラフ描画アルゴリズムにおける可視性の高さの基準 [8] を参考にプログラムに求められる可読性を定義した．

- 線が重ならない
- 線の間が適切な距離を保つ
- 線が滑らかである

3.2　文法の定義

BNF によって文法定義を行う．木文法 G=(S，N，F，R) は以下のように定められる．S は開始記号，N は非終端記号の集合，F は終端記号の集合，R を生成規則とする．このとき，N= { S，A，B_+，B_-，C_+，C_-，C^*_+，C^*_-，} ，F = { $a_\phi()$，$b_\phi(,\{\})$，$b_{\phi-}(,\{\})$，$a_+()$，$a_-()$，$a_2()$，$b_{++}\{,\}$，$b_{+-}\{,\}$，$b_{--}\{,\}$，$b_{-+}\{,\}$，$\beta_+\{\}$，$\beta_-\{\}$，$c_+(,)$，$c_-(,)$，l，n，cons(，)} とする．

流れの生成規則　R

S → $a_\phi(A^*)$ | $b_{\phi+}(B_+, \{C^*_-\})$ | $b_{\phi-}(B_-, \{C^*_+\})$

A → $a_+(B_+)$ | $a_-(B_-)$ | $a_2(C^*_+, C^*_-)$

A^* → n | cons(A，A^*)

B_+ → l | $b_{++}\{B_+, B_+\}$ | $b_{+-}(B_+, B_-)$ | $\beta_+\{C^*_+\}$

B_- → l | $b_{--}\{B_-, B_-\}$ | $b_{-+}(B_-, B_+)$ | $\beta_-\{C^*_-\}$

C_+ → $c_+(B_+, C^*_-)$

C_- → $c_-(B_-, C^*_+)$

C^*_+ → n | cons(C_+，C^*_+)

C^*_- → n | cons(C_-，C^*_-)

二次元多重領域上の流れは大域的に 3 つの基本パターン a_ϕ，$b_{\phi+}$，$b_{\phi-}$ からなる．a_ϕ は一様流を表す．$b_{\phi+}$，$b_{\phi-}$ はともに円盤状の流れで最外境界部をもち，それぞれ＋が反時計回り，－が時計回りの流れを表す．

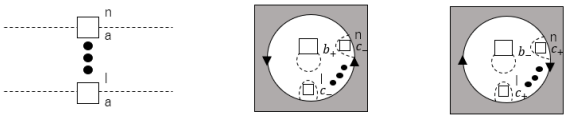

図 3.1　3 つの基本パターン　左から a_ϕ，$b_{\phi+}$，$b_{\phi-}$

ここで，$\square_T^L$ はホールである．T にはどの非終端記号から生成される木であるかという情報と流れの向きを表す．一方，L はホールが複数存在した場合, そのホールを区別するために利用するラベルである．また，文脈からホールが明らかな場合ラベルを省略することがある．そのほかにも，A 系の流れ，B 系の流れ，C 系の流れの構造が存在する．基本パターンに，これらの構造を生成規則に則り，組み合わせることで流線図を表現する．

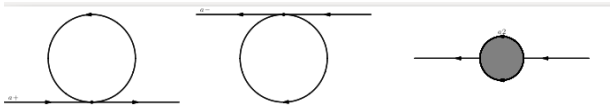

図 3.2　a 系の流れ　左から a_+，a_-，a_2

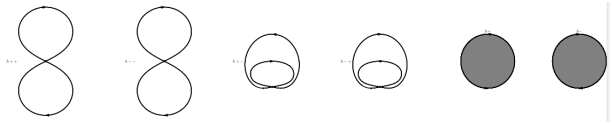

図 3.3　b 系の流れ　左から b_{++}，b_{--}，b_{+-}，b_{-+}，b_+，b_-

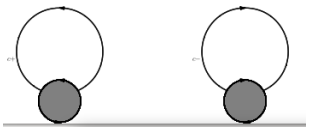

図 3.4　c 系の流れ　左から c_+，c_-

本研究では, 木表現を確実に図に自動変換することが目的であるため [3] で考えられた生成規則の円順列 {} については考えないものとし, 入力する文字列を容易にするため, ホール部分をそれぞれ置き換える．よって，実際に入力する文字列は例として以下のように入力する．

$b_{\phi+}(b_{++}\{\mathrm{l},\ \mathrm{l}\},\ \{\mathrm{n}\})$　→　　b0+(b++（l，l），(n))

3.3　アプローチ

本研究の目的の達成には，3.1 の定義を満たすトポロジーな流れの図を二次元上の画面に表示することを必須としている．アプローチとしては，コンピュータのディスプレイ上に作成した図を可視化し，画像ファイルとして保存することを可能にするプログラミング言語を本研究で使用した．私達は様々な分野での実績があり多様な環境での信頼性が高い Python とベクタ形式での描画を支援する Asyptote の 2 つプログラミング言語をアプローチ方法とした．

3.4 構文解析

本研究では Python での実装を行うため，Python 専用の構文解析ライブラリ PLY を利用し構文解析を行った．PLY とは，Python Lex-Yacc の略であり，Python 専用の構文解析ライブラリであり,lex モジュールと yacc モジュールからなる．lex. モジュールでは，入力されたテキストを正規表現により定義されたデータ構造に分解し，字句解析を行う．プログラムコードの概要として，字句の部分を tokens で定義し, 定義した字句の正規表現による具体的なルールを下の部分で定義していく．

```
tokens=(
  ' 定義する字句名',
  ...
)
t_定義した字句名=r' ルールの記述'
```

本研究では,3.2 によって定義された終端記号を字句として定義し, 終端記号の具体的なルールとして入力文字を表す．また, 改行は余白は無視する．例としてプログラムコードの一部を記述する．

```
import ply.lex as lex
tokens=(
  'CONS',
  'NIL',
  'A0',
  'LPAREN',
  'RPAREN',
  'COMMA',
  'A_PLUS',
  ...
)
t_ignore = '\t\n'
t_NIL = r'n'
t_A0 = r'a0'
t_LPAREN = r'\( '
t_RPAREN = r'\) '
t_COMMA =r','
t_A_PlUS = r'a\+'
...
```

その他にも，t_error() でエラー処理を行ったり，lex.lex() でオブジェクトの生成を行う．

一方，yacc. モジュールでは，lex モジュールによって定義された字句を読み込み，自由文法によって定義された構文を評価して構文解析を行い抽象構文木を作成する．プログラムコードの概要として，lex モジュールにより定義された字句を読み込み，読み込んだ字句からなる木構造を定義していく．

```
import ply.yacc as yacc
from lex モジュール import tokens
```

```
def  p_関数名 (p) :
    ' 文法規則'
    p[  ] = ···
```

それぞれの関数は一つの引数 p を受け取り，引数 p はルールと一致した字句の値を持っている．定義された引数は，連続した値になっており，左から順番に定義される．文法規則に則って木構造を生成していく．本研究では，yacc モジュールが LALR 法を用いて構文解析を行っているため，字句ごとにクラスを作成し，クラス内で抽象構文木の生成と同時に大きさの定義 4.1.2 を行った．例としてプログラムコードの一部を記述する．

```
import  ply.yacc  as  yacc
from  lex  import  tokens
import  sys
import  flow
def  p_exper_a0(p)  :
    ' s  :  A0  LPAREN  as  RPAREN'
    p[0]  =  flow.A0(p[3])
def  p_exper_b0_plus(p)  :
    ' s  :  B0_PLUS  LPAREN  b_plus  COMMA  LPAREN  cs_plus  RPAREN  RPAREN'
    p[0]  =  flow.B0_plus(p[3],p[6])
def  p_exper_b0_minus(p)  :
    ' s  :  B0_MINUS  LPAREN  b_minus  COMMA  LPAREN  cs_minus  RPAREN  RPAREN'
    p[0]  =  flow.B0_minus(p[3],p[6])
```

小文字で表記されている部分から枝が派生され，それぞれが非終端記号を表している．文法ルールが一致した場合，flow.py に存在する各クラスを呼び出し抽象構文木の生成，図の大きさの定義を行う．上記で記述したプログラムは，3.2 の開始記号 S から始まる文法定義を表しており，その他の非終端記号も同様にして記述していく．実際の流線の描画は flow.py の各クラスで描画していく．

3.4.1 デザインパターン

ソフトウェア設計者の経験から同じような問題は，典型的に同じパターンの解決策になることが発見され，それらのパターンをカタログ化したものをデザインパターンという．オブジェクト指向において，様々なプログラムで利用でき，目的に応じて 23 種の中から選択して扱うことができる．主にオブジェクト指向言語で再利用性が高いクラスやライブラリを作成する際に利用される．本研究では，デザインパターンの中のインタプリタパターンを取り扱う．インタプリタパターンについては，[9] を参考にした．インタプリタパターンは，形式的に記述された記号列を解析した結果に則って処理したい場合に利用されるデザインパターンである．目的に応じた言語を作成することで，通訳の意味を果たすプログラムを用意することにより，素早く処理することが可能になる．再帰的な構造を持ち，全ての要素に共通な処理を持つ抽象クラスを定義することで構造の変更を容易にしたり，見た目をシンプルにすることが可能である．本研究では，木文法より定義された生成規則によって，字句解析と構文解析を行い，構文木を作成する．そのため，構文木に則った処理を実現するのに最適なインタプリタパターンを利用した．また，作成するクラスが多いため，オーバーライドを利用することで機能の追加が容易になり，修正を減らすことができる．

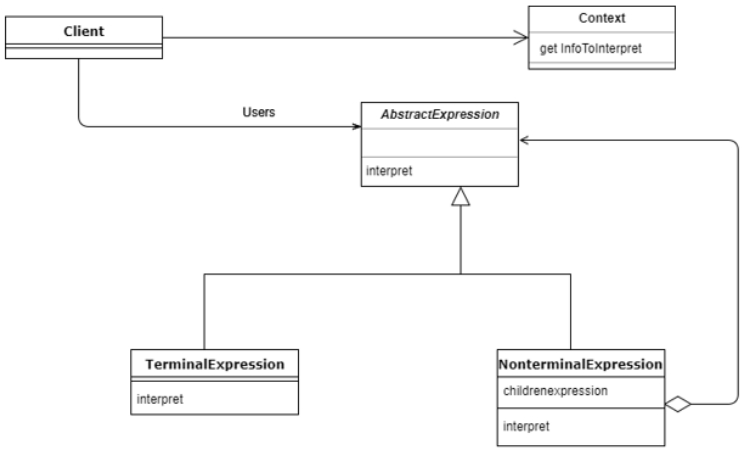

図 3.5　インタープリタパターン

3.5　Python での描画に使用するライブラリ

python では流線図を描画するにあたり，python 用のグラフ描画ライブラリである matplotlib を使用する．matplotlib はグラフ描画ライブラリであるが，棒グラフや折れ線グラフなどの一般的なグラフの描画だけでなく，平面上への関数・図形の描画や波のシミュレーションなど様々な描画を行うことができる．例えば，木表現 $b_{++}\{,\}$ を matplotlib を用いて作図しようとする場合，プログラム 3.1 のように記述することで図 3.6 が出力される (このプログラムでは，例を示すため明示的に円の中心と半径を示し描画を行った．また，プログラム簡略のため矢印と停留点を描画していない．)．このプログラムの 7 行目までは描画のための設定であり，描画領域として高さ・幅ともに 100 の正方形を用意している．また，draw_circ は円を描画する関数であり，描きたい円の中心の座標と半径を引数にとる．ここでは (x,y) 座標をそれぞれ (50,30)，(50,70) とする 2 つの円を描画することで $b_{++}\{,\}$ を表現した．描画した画像は savefig で保存することができる．この例では，EPS 形式で保存した．

プログラム 3.1　$b_{++}\{,\}$

```
1  import matplotlib.pyplot as plt
2
3  fig = plt.figure()
4  ax = fig.add_subplot(111,aspect='equal')
5  W,H = 100,100
6  ax.set_xlim([0,W])
7  ax.set_ylim([0,H])
8
9  def draw_circ(center,r):
10     circ=plt.Circle(center,r,ec="black",fill=False)
11     ax.add_patch(circ)
12
13 draw_circ((50,30),20)
14 draw_circ((50,70),20)
15
16 fig.savefig("b++.eps")
```

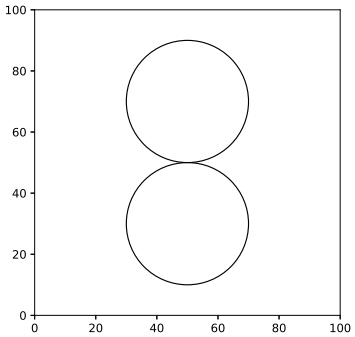

図 3.6　$b_{++}\{,\}$

3.6　離散的な点を結ぶスプライン補間

第 4 章

実装

4.1 Python による実装

4.1.1 実装するクラス図

本クラス図は 18 のクラスからなり，抽象クラスとして，Node クラスを定義し, 非終端クラスと終端クラスに共通するインターフェースを定義する．残りのクラスは，主に描画する流線パターンに対応している．また, 流線パターン以外にも存在し,Cons クラスによりリストの連接を表す．また，非終端クラスの集合｛A0，B0_plus，B0_minus，A_plus，A_minus，A2，Cons，B_plus_plus，B_plus_minus，Beta_plus，B_minus_minus，B_minus_plus，Beta_minus，C_plus，C_minus｝，終端クラスの集合｛Nil，Leaf｝とする．

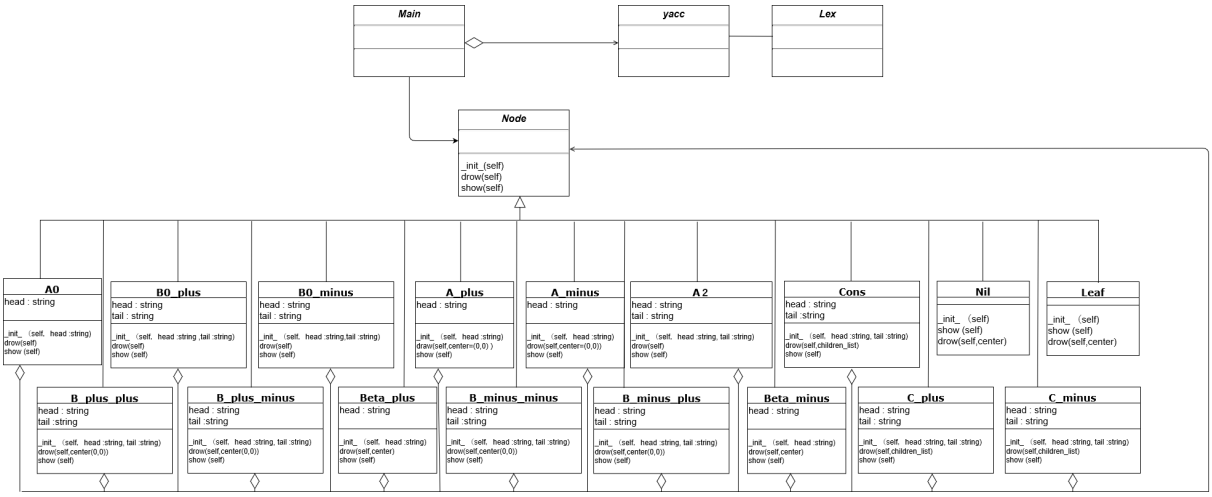

図 4.1　作成したクラス図

4.1.2 描画機能の実装：大きさの決定

Python で流線図を描画するにあたり，Python 用のライブラリである matplotlib を使用する．matplotlib はグラフ描画のためのライブラリであり，棒グラフや折れ線グラフなどの一般的なグラフを作図することができる．また，それ以外にも平面上への関数・図形の描画や波のシミュレーションなど様々な描画にも利用できる．

自動描画を実現するプログラムは木表現を与えるのみで動作することが求められ，私達が引数を与えて円を描くことはできない．また，子を持つ親を描画する際，子は親の描く流線の内部に描画される必要があるため，必然的に親の大きさは子の大きさよりも大きくなる．そのため，プログラムには描画した際の大きさを柔軟に

変更できる仕組みを組み込む必要がある．本研究では，この課題を解決するため木表現の葉から大きさを決定することにした．例として，$a_+(b_{++}(b_{++}(l,\,l),\,l),\,l))$ という木表現を挙げる．この木表現では，a_+ が b_{++} という子を持ち，その子である b_{++} は $(b_{++}(l,\,l),\,l),\,l))$ という 2 つの子を持っている親である．この場合，プログラムは $b_{++}(l,\,l)$ からその大きさを決める．続いて，$b_{++}(l,\,l)$ の親である $b_{++}(b_{++}(l,\,l),\,l)$ の大きさを決定する．ここで，親の b_{++} の大きさは，子の $b_{++}(l,\,l),\,l)$ を踏まえた大きさとなる．そして最後に，その大きさをもとに，a_+ の大きさが決定される．この流れを図 4.2 に示した．ここで，それぞれについて「大きさを決定する」としているのは，大きさを決定した時点では matplotlib のグラフ上に描画しないからである．これは，子から描画しようとしても，その時点ではグラフ中の位置を決定できないことによる．この例では，$b_{++}(l,\,l)$ の位置を先に決定してしまうと，その親である b_{++} の位置もそれに依存して決まってしまう．しかし，もし親の b_{++} がもう一つの子を持っていた場合，葉から描画していく仕様上もう一つの子も既に決められた位置を持つことになる．ここで，子の位置を決定する時点で他の流線との位置関係を決定できる仕組みがあり，2 つの子の位置の整合性が取れているのであれば子から順に描画することもできるが，子の情報だけでは親との位置関係は分からないためこの仕組みは作成できない．そのため，実際にグラフ上に描画する前に根までのすべての大きさを決定する処理を行い，次に根の位置を決定し順に子を描画していく方法を取った．

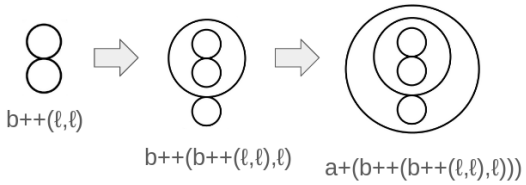

図 4.2　$a_+(b_{++}(b_{++}(l,\,l),\,l))$ 大きさ決定のイメージ

4.1.3　描画機能の実装：流線の描画

matplotlib では，円や長方形などの図形をもともと用意されている関数で呼び出したりできるが，これらは決められた形にしかならない．流線を描画するにあたり，一部のトポロジーは円などの固定された図形でも表現可能だが，β 系を含む流線はそれらで表すことができず，時々に合わせて形を変えることが求められる．そこで，本研究ではこれらの β 系を含む図形を点でモデル化することにした．例として，a_+ が子に β 系を持っていると想定した場合の流線を図 4.3 に示す．ただ，これは製作途中の仮のものであるため，その子の大きさの円を仮に与え描画している．この図形は，点 5 つから構成されている．子の専有領域の接合部である 2 点と最右点・最左点・最下点の 3 点である．この 5 点を、スプライン補間で補間することにより流線を表現した．

図 4.3　β 系を持つ図形の作図

4.1.4　描画機能の実装：個々の描画アルゴリズム

A0 の実装
B0 の実装
A+ の実装
A2 の実装
B+ の実装
Beta の実装
C+ の実装

4.1.5　描画の限界

4.2　Asymptote による実装

本研究での目的達成の成果物は明確には定めておらず，一意に決まるものではない．Asymptote による実装では，与えられた木表現に対して 3.1 の定義を満たす図を作成するためのライブラリを作成することで本研究の目的の達成したと言えることとする．

4.2.1　Asymptote による実行方法

Asymptote ではプログラムを実行する方法として，Ubuntu の端末や Windows のコマンドプロンプト上に直接コードを記述する interactive mode(対話モード) という方法が存在する．そこでライブラリを読み込み，画面上に表示させたい流線図に対応する関数を記述することで，画面上にトポロジー化された流線図を表示させる．

4.2.2　提供するライブラリ

ライブラリ内の流線図には，図に配置の基準となる点や大きさなど，利用者が指定しなければならないパラメータが必ず付与されている．3.1 の定義を満たす図を作図するためには，ずべての流線図の特徴とコードを把握する必要がある。

パラメータのデータ型

- pair　x 座標と y 座標の二次元座標における複素数を表すデータ型であり、本研究では図の基準点として扱う。例えば実部 x に 0、虚部 y に 0 を代入すると（0，0）を基準点として作図される。
- real 実数を表すパラメータであり、本研究では渦心円の大きさとして扱う。a+、a-の図では渦心円の半径を示している。すなわち、渦心円の大きさは real 型に与えられた実数に比例する。渦心円の中には他の流線図が入る場合もため、その場合は中の流線図より大きい値を real 型に与えなければ、3.1 の定義を満たない図となるため作図するたびに調節する必要性がある。
- bool3 値として true、default、false の 3 つを与えることができる。本研究では a 系、b 系、 c 系の流線図を作成するライブラリの作成を目標としており、それぞれの下付き文字の種類は＋（プラス）、ー（マイナス）、2 の最大 3 種類である。bool3 型はそれらを区別するために採用したデータ型である。似たデータ型で bool 型が存在するが、bool 型は true、false の 2 種類しか与えることができない。すべての流線図を網羅するためには 3 種類の値を与えられる bool3 型を使用する必要があった。本研究では bool3 に true が与えられると a+ や b+ などの渦心円が反時計回りの軌道での流れを示すプラスの流線図が表現される。false が与えられたのならば、a-や b-などの渦心円が時計回りの軌道での流れを示すマイナスの流線図が表現される。また、default が与えられたならば a2 などの 2 の流線図が表現される。

関数

- draw 描画するための線を描く命令をする関数である．draw((x1,y1)–(x2,y2)) により、座標 (x1,y1) から (x2,y2) までの直線を描くことができる．またラベルや矢印を付与したり線の色を変更したりする特性を持つことも可能である．本研究では、矢印を付与する arrow と円を描く circle の特性を持つ draw 関数を使用した．
- fill 指定した場所に、指定した色を塗りつぶす命令をする関数である。本研究では障害物を灰色で塗りつぶすことで、流線との違いを明確にしている。障害物は円状にしているので circle 関数と共に使用される。
- arrow 本研究では矢印を流線上に表記することで、流線の軌道の進む向きを表現している。arrow 関数はその際に使用する矢印を表現する関数である．draw 関数の内部に arrow 関数を与えることで流線上に矢印付与することができ、流線図の拡大縮小の際にも整合性を保つことができる。矢印の場所は draw 関数で描く線分の始点、中間点、終点の 3 種類である。

図 4.4　左から BeginArrow、MidArrow、EndArrow

この 3 つの場所以外に書く場合は Relative 関数をともに使用して描く。また、矢印の種類は変更することも可能であり、4 種類の矢印が存在する。

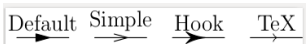

図 4.5　左から DefaultaHead、SimpleHead、HookHead、TeXHead

SimpleHead や TeXHead などの微細な矢印では拡大縮小の際に矢印が確認できなくなり流線の軌道の進む向きが判断できなくなる恐れがある。本研究では、HookHead を使用して流線の完全性を保てるようにする。

- circle 円を描く命令する関数である。中心点となる座標と半径となる大きさを与えることで任意の円を描くことができる。本研究では流線の渦円心と障害物を表現する際に使用される。
- dot 点に近い一定の大きさの円を与えられた座標点に描く命令をする関数である。本研究では淀み点を表現するために dot 関数を使用する。淀み点は黒丸なので fill 関数と circle 関数を用いて表現することも可能であるが、dot 関数を用いることで簡潔に表現できるので使用した。また、大きさが一定であるため流線の拡大縮小の際にも完全性を保つ流線を描くことが可能である。
- Relative
- scale ・scale の特徴・scale の役割・scale

4.2.3　a+ と a-

【ここにソースコード】

a+ と a-の特徴

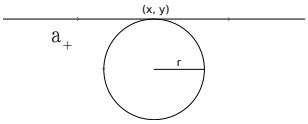

図 4.6　a+

【a-の図】・a+ と a-の特徴【説明は参考文献から持ってくる】・図における基準点や半径の説明・渦円心の中に入る流線図

4.2.4　b 系

【ここにソースコード】

b 系の特徴

【図】

・特徴

・２つの bool3

・渦円心の中に入る流線図

4.2.5　構造体

・構造体での表現による利点　・オブジェクト指向的な設計　・隠蔽

4.2.6　クラス図との対応

・クラス図に対応している場所や有無

4.2.7　描画の限界

・線が潰れる

4.2.8　まとめ

・入力を完全な木表現？で作図することはできず　・原因

第 5 章

おわりに

5.1 Python

作成中のプログラムは，現在再現できる a 系と b 系の組み合わせの範囲内では 3.1 により定められた条件を満たす図を製図できる．また，まだ再現できない系の流線に対しても葉から大きさを決定し根から描画する考え方は適用可能であると考える．しかし，一部の流れにはループが存在し，その部分の要素は無限に増やすことが可能である．そのため，要素数が無限に増えても柔軟に生成される図を変更できることが求められる．現在，これを解決するためグラフ描画アルゴリズムを利用を検討している. グラフ描画アルゴリズムを用いることで，文献 [8]，[11] にあるように柔軟に図の変更を行うことができるのではないかと考えている．具体的には C 系の中ので発生する流れ同士の距離感を一定に保つことなどを考えているが，まだ実装には至っていない．今後は，これらの描画手法を実装し，また，作成したプログラムの有用性に関しての検証を行っていきたい．

5.2 Asymptote

Asymptote で作成したプログラムも Python と同等な理由で，c 系を組み合わせ可能の段階には至っていない．また，本研究の目的である入力を完全な木文法での描画は現状実装できなかった．今後は，本研究の目的達成のために改良を施していきたい．

参考文献

[1] 荒井迅：トポロジカルな流れ構造の理解へ向けて，ながれ，Vol.33，No.1，pp.23–28 (2014).

[2] Yokoyama, T. and Sakajo, T.: Word representation of streamline topologies for structurally stable vortex flows in multiply connected domains. *Proc. Roy. Soc. A*,Vol.469,No.2150,pp.1–18 (2013).

[3] 加藤舞，内藤綾香，横山哲郎，横山知郎：円盤上の非圧縮流の反転の解析, 情報処理学会　第 81 回全国大会講演論文集，Vol.1900，pp.319–120(2019).

[4] 横山知郎，坂上貴之，澤村陽一：二次元多重連結領域内における構造安定な非圧縮流れの文字列表現アルゴリズム，数理解析研究所講究録，Vol.J101-D．pp.11–25(2014).

[5] 横山哲郎，横山知郎：ハミルトン曲面流に対応する語の列挙アルゴリズム，電子情報通信学会論文誌 D，Vol.100，No.10，pp.892–894(2017).

[6] 横山哲郎，横山知郎：ハミルトン曲面流に対応する流れの向きを考慮した極大語の列挙アルゴリズム，電子情報通信学会論文誌 D，Vol.101，No.8，pp.1220–1222(2018).

[7] 加藤舞，内藤綾香：多重連結領域上の安定非圧縮流の解析，南山大学 2018 年度卒業論文 (2019).

[8] 土井淳，伊藤貴之：力学モデルを用いた階層型グラフデータ画面配置手法の改良手法とウェブサイト視覚化への応用，芸術科学会論文誌，Vol.3，No.4，pp.250–263(2004).

[9] 結城浩：Java 言語で学ぶデザインパターン入門，SB クリエイティブ (2001).

¡¡¡¡¡¡¡ HEAD =======

[10] Brøns, M. Jakobsen, B. Niss, K. Bisgaard, A. and Voigt, K, L.: Streamline topology in the near wake of a circular cylinder at moderate reynolds numbers, *Journal of Fluid Mechanics*,Vol.584, pp.23–43 (2007).

¿¿¿¿¿¿¿ ea82b7ff0a19989d6216cdece582f504b1e37876

[11] 谷崎正明，丸山貴志子，嶋田茂：デフォルメマップ生成のための道路形状正規化モデルとそのシステム評価，電子情報通信学会論文誌. A，Vol.87，No.1，pp.108–119(2004).

付録 A

プログラムリスト

付録 B

実行例

実行例を示す。

```
Hello, world
```