卒業論文

Rubyにおける数式処理アプリケーションのwrapperの開発

関西学院大学理工学部

情報科学科 西谷研究室

27017607 堀川恭平

2021年3月

概 要

本研究では，普段 Maple を用いて行っている数式処理やグラフのプロットを Ruby スクリプトから呼び出すことを目標とした．利用するソフトウェアとして，Maple と Sagemath を比較して，ライセンスの形態や記述のシンプルさから，本研究では，Sagemath を用いて開発していくこととした．

結果として，

- Sagemath との通信部を含む Sage クラス．
- 方程式計算，微分などの数式処理を行う Algebra クラス．
- 画像のプロットを行う Plot クラス．

を開発した．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景

西谷研究室では，多くの研究を第一原理計算の大規模計算のためのソフトウェア VASP，まつもとゆきひろにより開発されたオブジェクト指向スクリプト言語 Ruby，数式処理，数値計算，グラフ作成などを行うソフトウェア Maple を用いて進めている．これは，Ruby による数式処理の計算のライブラリが整っていないことから，それぞれの利点を生かした運用をするためである．図 1.1に，西谷研究室での流れの一例を示した．

はじめに，Ruby スクリプト内から VASP を呼び出し，計算結果が格納されたファイルを得る．このファイルは，可読性が低いため，さらに Ruby スクリプトに与えて，再利用しやすいように加工する．その加工された結果を用いて，Maple スクリプトを用いて，2 次元フィッティングをして，エネルギーの最安定値を求める．

このような流れで研究を進めるが，Maple は，Ruby と記法が大きく異なるため，学習コストが高く，Maple の習得に研究のリソースが割かれてしまうという場面が数多く見られた．

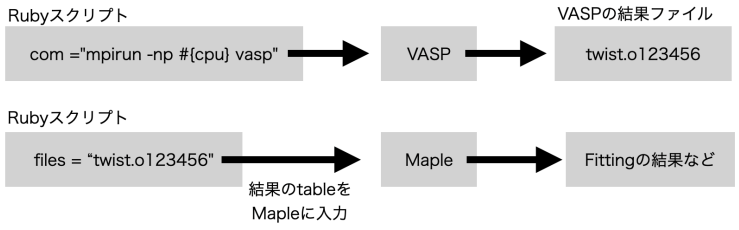

図 1.1: 現在の研究の流れの一例.

1.2 研究の目標

図 1.2の流れを実装する．従来 Maple を用いて扱っていた数式処理，グラフのプロットを Ruby のスクリプト内より手軽に扱うことで，研究のリソースが割かれることを防ぎ，他の研究に時間を費やせる環境の構築を本研究の目標とする．

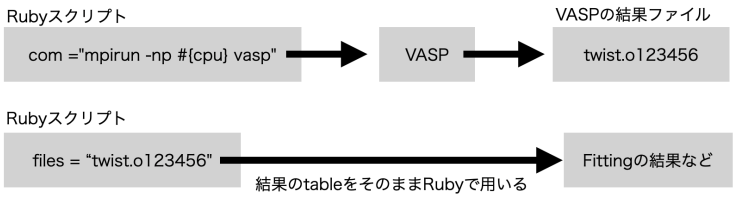

図 1.2: 目標とする研究の流れの一例.

第2章　手法

大きく分けて2つの手法を考えた.

2.1　手法の比較

- Mapleを用いる方法.
- 数学の幅広い処理を扱うソフトウェアであるSagemathを用いる方法である.

結果として本研究では，Sagemathを用いることとした．以下の表2.1に，MapleとSagemathの比較を示す.

表 2.1: MapleとSagemathの比較.

	Maple	Sagemath
ライセンス	有料	無料，オープンソース
出力の再利用性	工夫が必要	再利用可能
画像ファイルの出力	複雑なコマンド	理解しやすいコマンド

Sagemathを用いる決め手となったのは，誰でも無料で利用できるという点である．関西学院の学生は，Mapleのライセンスを用いることができるが，最終的に，本研究が様々な場面で利用されることとなったとき，有料では都合が悪い.

次に比較したのは，出力の再利用性である．研究では，方程式の解を用いて，新たな計算をするといったことがある．そのため，図2.1のようなMapleの出力では，改行や空白が多いことから，視認性は良いが，再利用性が低いと言える．それと比較して，図2.2では，改行や空白がなく，再利用しやすい形であるといえる.

また，目標の一つである画像のプロットにおいて，MapleではCLIで複雑なコマンドを用いる必要があるため，開発が難航した．それに対して，Sagemathは，CLIから容易に画像の出力ができた.

```
> solve(2*y-(x-1)^2=2,y);
                                    2
                          1/2 x  - x + 3/2
```

図 2.1: Maple で方程式を解く様子.

```
sage: solve([2*y-(x-1)^2==2],y)
[y == 1/2*x^2 - x + 3/2]
```

図 2.2: Sagemath で方程式を解く様子.

これらの理由から本研究では，Sagemath を利用することとした.

2.2 開発の方針

Maple,Sagemath ともに開発の方針は同じなので，先に示す.

2.2.1 Ruby の PTY モジュール

Ruby には，疑似端末 (Pseudo tTY) を扱うモジュールである PTY と呼ばれるモジュールがある．この PTY に対して，Maple や Sagemath の実行ファイルとコマンドを渡すことで，Maple や Sagemath からの出力を受け取ることができる.

```
getpty(command) -> [IO, IO, Integer]
```

ここで，command は，擬似 tty 上で実行するコマンドを表す．返り値は，3 つの要素からなる配列である．最初の要素は擬似 tty から読み出すための IO オブジェクト，2 番目の要素は書きこむための IO オブジェクト，3 番目の要素は子プロセスのプロセス ID である [1].

この PTY モジュールを利用し，開発を進めていく.

2.3 Maple での開発

以下に Maple を用いた開発の記録を残す [2].

```
require "pty"
require "expect"
require "timeout"
class Maple
  def initialize(command)
    @maple = "/Library/Frameworks/Maple.framework/Versions/Current/bin/maple"
    @command = command
    @num = @command.count(";")
  end
  def get_res
    PTY.getpty(@maple) do |i, o|
      begin
Timeout.timeout(1) do
  loop { i.getc }
end
      rescue Timeout::Error
      end
      o.puts @command
      res = []
      finished = false
      while !finished
i.expect(/>|(.*)\n/) do |m|
  if m[1].nil?
    finished = true
  end
  res << m[0].gsub!(/\e\[;*\d*m\s*/, "")
  # p res << m[0]
end
      end
```

```
      return res[-3].gsub(/\"/, "") unless res[-3].nil?
    end
  end
end
```

はじめに，必要となるモジュールを require している．Maple クラスの initialize では，Maple の実行ファイルを指定し，実行するコマンドを引数として受け取り，インスタンス変数に格納する．

コマンドを Maple で実行して結果を求める，get_res 関数では，tty から読み出すための IO オブジェクトを i, 書きこむための IO オブジェクトを o として，ブロック引数をとっている．Maple を CLI で起動すると，以下の図 2.3 のようなアスキーアートが描画される．

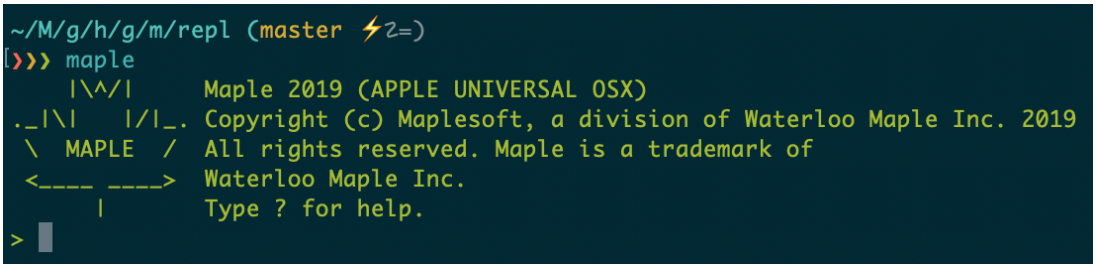

図 2.3: Maple 起動時のアスキーアート．

これを出力として受け取らないように，Timeout モジュールを用いて，1 秒間，IO オブジェクトのメソッドである，getc で 1 文字ずつ読み進めていく [4]．最後に，o に対して実行するコマンドを渡して結果を得る．ただ，この結果には，図 2.4のように ANSI エスケープシーケンスが多数含まれていて，可読性が低く，結果の再利用ができない．

図 2.4: 出力に ANSI エスケープシーケンスが含まれる様子．

そこで，gsub メソッドを用いて，余分なエスケープシーケンスを除去して，整形する．

以上が，Ruby から Maple を呼び出す際に基幹となる部分の開発である．

第3章　結果

3.1　Sagemathでの開発

前述の方針を元にSagemathを用いた開発を行った．今回の開発では，Rubyのクラスの概念を有効に利用した．クラスとは，オブジェクト指向における設計図の役割を担うものであり，クラスからインスタンスを作成してそのインスタンスで処理を行う．

メリットとして，

- インスタンスにはそれぞれのインスタンス変数という変数が存在するため，一貫性が生まれ，それぞれの独立を守ることができる．
- 細かいパーツに分けることで，保守や追加の機能開発の際，バグが起きづらい．

といったことが挙げられる．

3.1.1　Sagemathのクラス構成

以下の，図3.1は基幹となるSageクラス，プロットに関する機能を持ったPlotクラス，数式処理の機能を持ったAlgebraクラスの継承関係を表す．

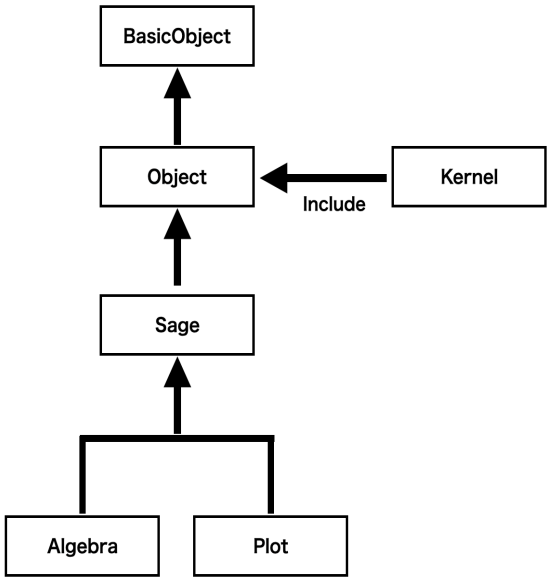

図 3.1: SageClass の継承関係.

図 3.1のような継承関係を持つため，Plot クラス，Algebra クラスからも，Sage クラスが持つメソッドを利用することができる.

3.1.2 Sage クラス

Sage クラスは，図 3.2のメソッドを持っている．それぞれのメソッドについて解説する.

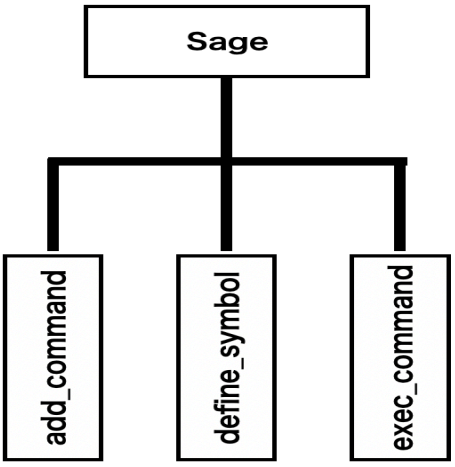

図 3.2: Sage クラスの概要.

1. initialize

```
def initialize
    @define_command = Array.new
    @command_list = Array.new
    @command = ""
    @res = Array.new
    @flag = true
end
```

Ruby のクラスにおいて，initialize という名前のメソッドは，インスタンスが作成されたときに自動で呼ばれる．ここで，@から始まる変数はインスタンス変数である．各インスタンス変数は，以下の表 3.1が示す型をとり，それぞれの役割を果たす．

表 3.1: Sage クラスの変数.

@define_command	配列	symbol として使用する文字の初期化を格納する
@command_list	配列	各メソッドが呼ばれるたびに命令を格納する
@command	文字列	@command_list を; で繋いだ Sagemath に送る最終的な命令を格納する
@res	配列	Sagemath から返ってきた結果を格納する
@flag	真偽値	返ってきた結果を return するかを決める

2. define_symbol

define_symbol メソッドでは，文字を変数ではなくシンボルとして扱うための設定を行う．

ソースコードは以下の通りである．引数に可変長引数を採用することで，一度に複数個のシンボルを定義することが可能となった．可変長引数は配列の形で渡されるため，join メソッドを用いて，”,”区切りで結合させて使用している．

```
# ソースコード
def define_symbol(*symbol)
    symbol = symbol.join(",")
```

```
    @define_command << "#{symbol}=var('#{symbol}')"
end
```

使用例は以下の通りである．これで y,z をシンボルとして扱うことができる．

```
# 使用例
sage = Sage.new()
sage.define_symbol("y", "z")
```

3. add_command

add_command メソッドでは，引数をそのまま@command_list に加える．今回実装していない Sagemath のメソッドも add_command を用いることで利用することができる．

```
# ソースコード
def add_command(com)
    @command_list << com
end

# 使用例
sage = Sage.new()
sage.add_command("fit = find_fit(data, model, solution_dict=True)")
```

4. exec_command

exec_command メソッドは実際に Sagemath と通信する部分である．基本的にソースコードは，2.3 で解説した Maple での開発と同じである．異なる点として，結果を格納する変数@res の取り出し方がある．ここでは，"\e["というシーケンスに着目して，split メソッドで分割し，match メソッドと正規表現をもちいてキャプチャした．

```
# ソースコード
```

```
if @flag
  @res[-1].split("\e[")[-1].match(/0m(.*)\r\n/)
  return $1
else
  return 0
end
```

3.1.3 Algebra クラス

Algebra クラスは図 3.3のメソッドを持っている．特徴的なメソッドについて解説する [5].

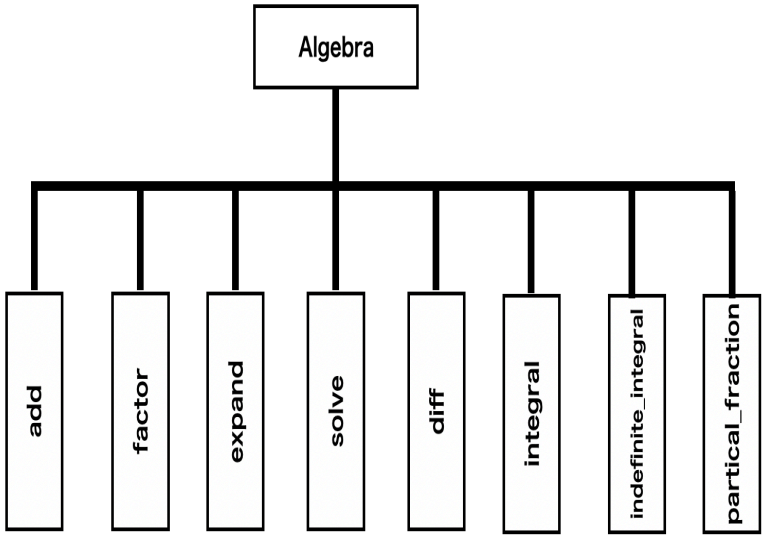

図 3.3: Algebra クラスの概要.

1. solve

   solve は方程式を解くメソッドである．入力を工夫することで連立方程式も解くこともできる．Sagemath では，方程式の左辺と右辺を"=="で繋ぐが，一般的な方程式と乖離しているので，gsub メソッドを用いることで"="の入力を受け付けることができる．

```
# ソースコード
def solve(f, target)
  f.gsub!("=", "==")
  @command_list << "solve([#{f}],#{target})"
end

# 使用例
algebra = Algebra.new()
algebra.define_symbol("y")
algebra.solve("x+y=6, x-y=4","x,y")
```

2. diff

   diff メソッドは微分の計算を行う．引数に整数を指定して，複数回微分に対応した．引数がなかったら一回微分となるように，Ruby のデフォルト引数を採用している．

```
# ソースコード
def diff(f, target, num = 1)
  @command_list << "diff(#{f},#{target},#{num})"
end

# 使用例
algebra = Algebra.new()
algebra.define_symbol("y")
algebra.diff("x^2 + 17*y^2", "x", 2)
```

3. integral，indefinite_integral

   integral メソッドは定積分，indefinite_integral メソッドは不定積分を計算する．

```
# ソースコード
def integral(f, target)
  @command_list << "integral(#{f},#{target})"
```

```
end
def indefinite_integral(f, target, from, to)
  @command_list << "integral(#{f},#{target},#{from},#{to})"
end

# 使用例
algebra = Algebra.new()
algebra.indefinite_integral("x/(x^2+1)", "x", 0, 1)
```

4. partial_fraction

   partial_fraction は部分分数分解を行うメソッドである.

```
# ソースコード
def partial_fraction(f, x)
  @command_list << "f=#{f}"
  @command_list << "f.partial_fraction(#{x})"
end

# 使用例
algebra = Algebra.new()
algebra.partial_fraction("1/((1+x)*(x-1))", "x")
```

5. Algebra クラスの使用例

   ここでは，VASP 計算で得られた値から，最安定値を求めていく.

```
# ソースコード
require "./Algebra.rb"

class FindEq < Algebra
  def initialize
    super
```

```
  end

  def fit(data)
    define_symbol("a0,a1,a2")
    add_command("data=#{data}")
    add_command("model(x) = a0+a1*x+a2*x^2")
    add_command("fit = find_fit(data, model)")
    add_command("f_fit(x) = model.subs(fit)")
    add_command("f_fit(x)")
    return exec_command()
  end

  def find(data)
    diff(fit(data), "x")
    diff = exec_command()
    solve(diff, "x")
    exec_command().match(/== \((.*)\)\]/)
    add_command("n(#{$1})")
    exec_command()
  end
end

data = [[-0.04, -117.2680211429], [-0.02, -118.9324825714],
 [0, -119.4657688571], [0.02, -119.0653597143],
 [0.04, -117.9490060000], [0.06, -116.2624700000]]
algebra = FindEq.new()
algebra.find(data)
```

これは,

(a) VASP 計算の結果を data として，それにフィットする曲線を求める.

(b) diff メソッドを用いて，微分し曲線の傾きを求める．

(c) solve メソッドを用いて，傾き=0，すなわち最小値を求めている．

結果として，x=0.00425965100073602 となり，この値で，最小値を取る．これは，以下の図 3.4からも正しいことが読み取れる．

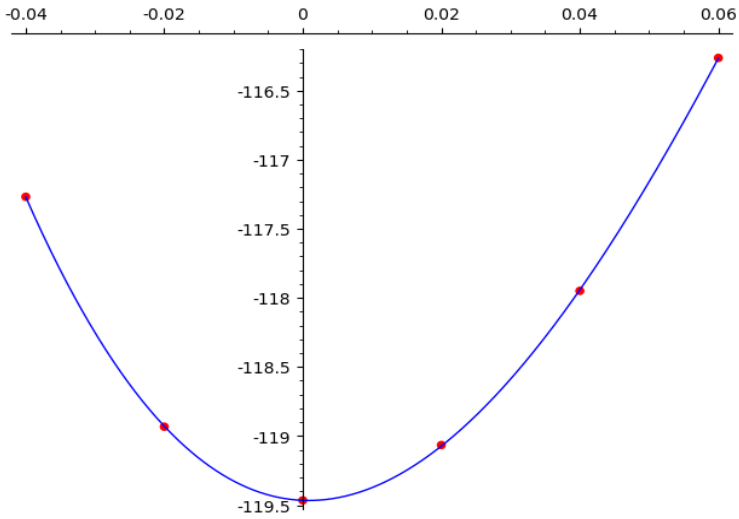

図 3.4: Algebra クラスの使用例．

3.1.4 Plot クラス

Plot クラスは図 3.5のメソッドを持っている．特徴的なメソッドについて解説する．また，Plot クラスでは，

- 引数にオプションを指定してプロットの挙動を制御したい．
- 引数を指定しないこともできる．
- 引数の順番を考慮しない．

といった理由からキーワード引数を多く採用している．

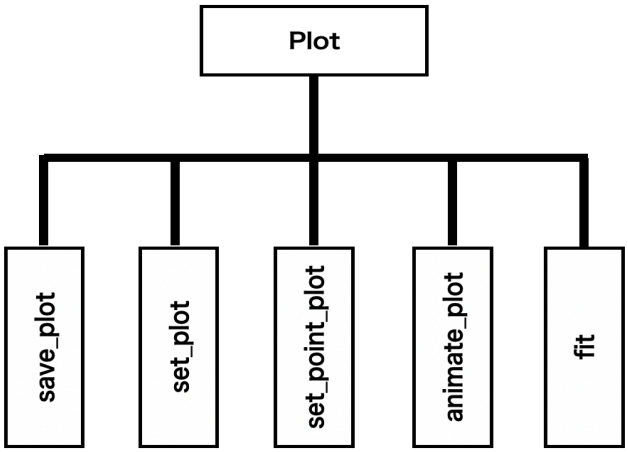

図 3.5: Plot クラスの概要.

1. save_plot

   save_plot メソッドでは，プロットした画像を保存する．前述した通り，容易に保存ができることが，Sagemath を採用した理由にもなっている．後述の set_plot や，set_point_plot と合わせて使用する．

```
# ソースコード
def save_plot(function, plotname: "plot", extension: "png")
  @command_list << "#{function}.save('#{plotname}.#{extension}')"
  @flag = false
end
```

2. set_plot

   set_plot メソッドでは，プロットをするための準備をする．キーワード引数として，プロットの範囲を指定する，xmin,xmax を取る．

```
# ソースコード
def set_plot(function, xmin: -5, xmax: 5)
  @command_list << "plot=plot(#{function}, (x,#{xmin},#{xmax}))"
end
```

```
# 使用例
plot = Plot.new()
plot.set_plot("3*x^2 + -7*x + 23")
plot.save_plot()
plot.exec_command
```

以下の図 3.6は，前述の save_plot メソッドと合わせて使った際の出力された画像である．

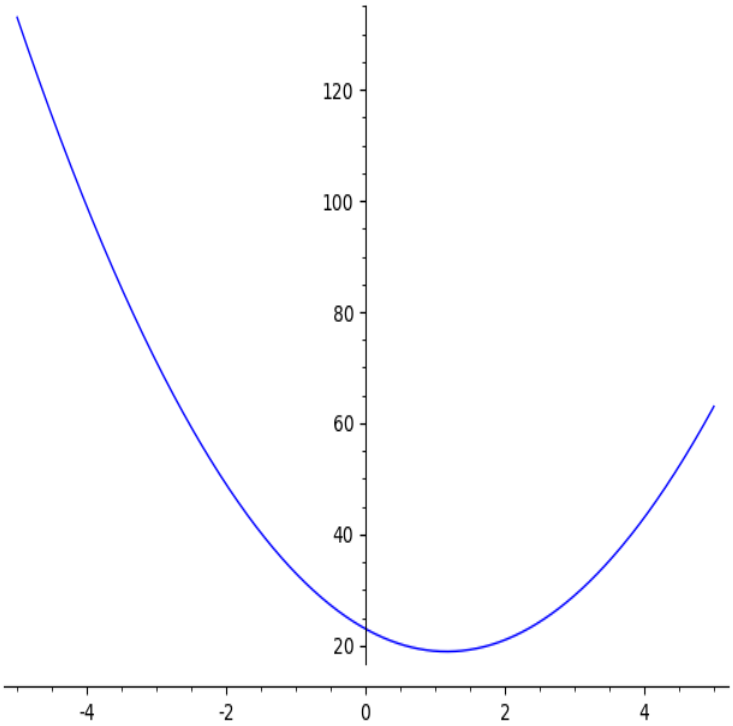

図 3.6: plot の例．

3. set_point_plot

set_point_plot メソッドは，引数として座標の配列の入った配列を渡すことで，点の描画をする．

```
# ソースコード
def set_point_plot(array, color: "red", size: 30)
  @command_list << "point=list_plot(#{array},color='#{color}',pointsize=#{size})"
end
```

```
# 使用例
array = [[1, 2], [2, 5], [6, 3], [7, -1]]
plot = Plot.new()
plot.set_point_plot(array)
plot.save_plot(function: "point")
plot.exec_command
```

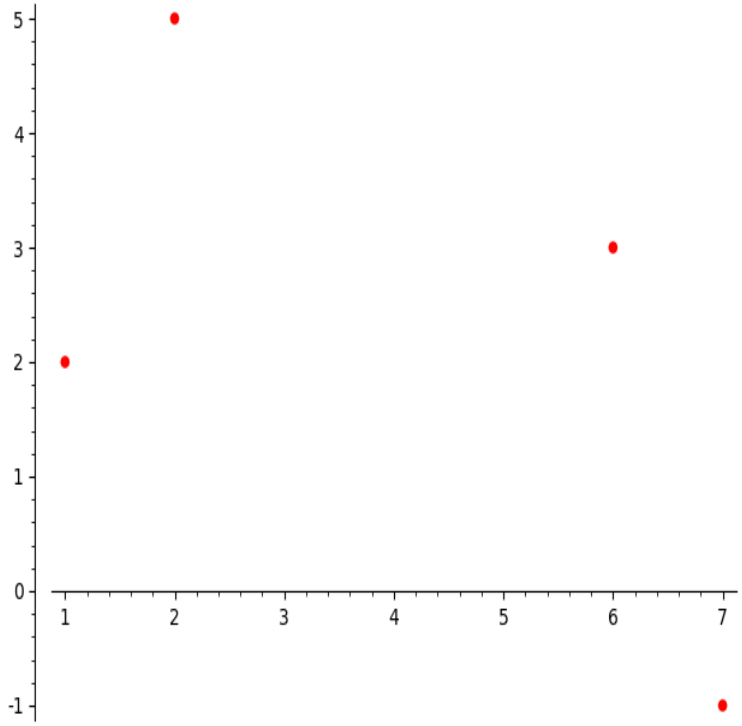

図 3.7: point plot の例.

4. fit

fit メソッドは，与えられた点にフィットする近似曲線を描画する [6]．実際に西谷研究室で，物理における数値解析にて頻繁に使用されている fitting と呼ばれる作業をメソッド化した．今回実装していない Sagemath のメソッドをいくつか使用しているため，add_command メソッドを用いて，直接記述することで実装している，ただ，この部分はユーザーが直接コードを書くことはないので，学習コストの増加を防いでいる．

フィッティングの手順として，

- モデルを定義する．今回の場合は，3 次関数をモデルとしている．
- Sagemath の find_fit を使用し，モデルにフィットする，w0,w1,w2,w3 を求める．
- モデルに決定した w0~w3 を代入する．

```
# ソースコード
def fit(array, xmin: -1, xmax: 1, plotname: "plot",
point_color: "red", curve_color: "blue")
  set_point_plot(array, color: point_color)
  define_symbol("w0", "w1", "w2", "w3")
  add_command("model(x) = w0 + w1*x + w2*x^2 + w3*x^3")
  add_command("data = #{array}")
  add_command("fit = find_fit(data, model, solution_dict=True)")
  add_command("f_fit(x) = model.subs(fit)")
  add_command("fit_plot = plot(f_fit, [x, #{xmin}, #{xmax}],
  rgbcolor = '#{curve_color}')")
  add_command("p = fit_plot + point")
  save_plot(function: "p", plotname: plotname)
end

# 使用例 1
array = [[1, 2], [2, 5], [6, 3], [7, -1]]
plot = Plot.new()
plot.fit(array, xmin: 0, xmax: 10)
plot.exec_command
```

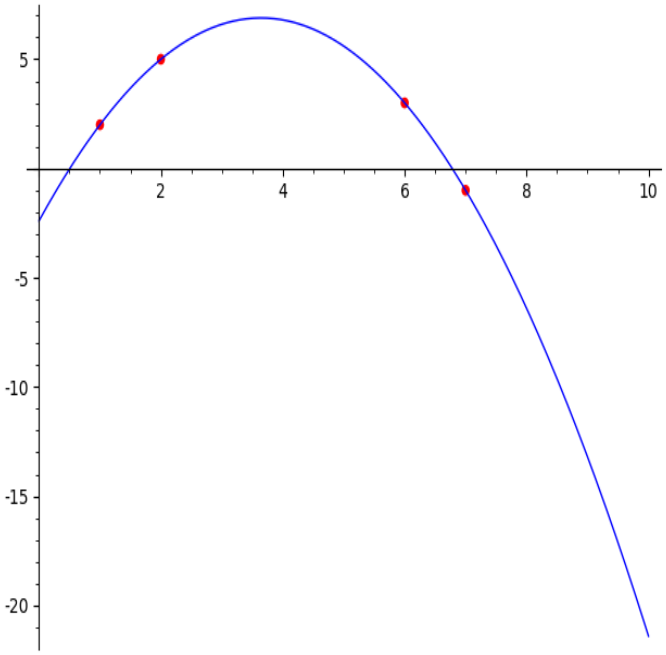

図 3.8: フィッティングの例 1.

```
# 使用例 2
array = [[0.0, -119.95449], [-0.2, -119.94399], [-0.1, -119.95251],
[0.1, -119.95253], [0.2, -119.94389]]
plot = Plot.new()
plot.fit(array, xmin: -0.2, xmax: 0.2)
plot.exec_command
```

図 3.8と図 3.9は，実際にフィッティングを適用した例である．図 3.8では，4 点全てにフィットする関数を描画できている．図 3.9では，少し点からずれてしまっているが，差が最小となるような曲線を描画している．これを用いることで，図 3.9では，x=0 の時，y=-119.955 で最小値を取ることが確認できる．

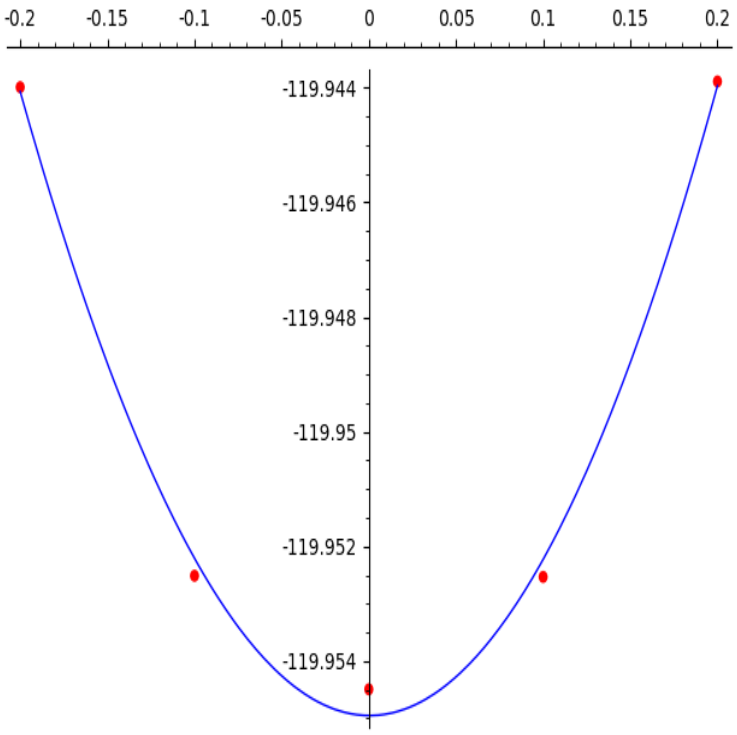

図 3.9: フィッティングの例 2.

第4章　結論

今回の開発では，Sagemath を Ruby の PTY モジュールを活用することで，当初の目標としていた数式処理とグラフのプロットを実現することができた．ただ，開発の途中で，Maple から Sagemath への方針転換したこともあり，数多くの機能を実装することはできなかった．Ruby の苦手とするこれらの処理を Ruby スクリプト内から呼び出すことで，学習コストの高い Maple を学ぶ過程をスキップして，研究を円滑に進めることができるようになった．

4.1　今後の展望

4.1.1　実行部分の改善

現在の実行部分の実装は，コマンドを送ってから一定の時間待つことによって，出力を出し切り，それを取得している．そこで生じる問題が，処理によってかかる時間が異なることである．やはり画像の出力には時間がかかるため，画像の出力に合わせて数式の処理を行うと，冗長な処理となってしまう．そこで，数式処理なのか，画像の出力なのかを判別し，待機時間を可変とする仕組みを追加していく．

4.1.2　実際の研究での使用

実際の研究の中で，SageRuby を使用していきたいと考える．そうすることによって，まだ実装されていないが必要な機能や改善が必要な機能が浮き彫りになり，SageRuby をもっと発展させていくことができると考える．また，自分とは違う環境で使用することによって，予期しないバグの発見に繋がり，安定した機能を提供できると考える．

4.1.3　テストの作成

テストとは，開発したプログラムが想定通りの動きをするかどうかを確認するためのものである．今後，追加の機能を開発していくにあたって，既存の機能に影響を与えてしまわないかを確認する．Ruby の提供する，UnitTest を用いて，与えた入力に対して，正しい出力を得ることができているかを自動で確認する．

4.1.4　gem としての提供

実際に研究で使用するにあたって，SageRuby を配布する手段として，RubyGems を用いていこうと考える．メリットとして，

- プログラムの配布が容易である．
- バージョン管理が容易である．
- ダウンロード数がモチベーションに繋がる．

といったことが挙げられる．

謝辞

本論文の作成にあたり，多くの方々にご支援いただきました．
先ず，貴重なご指導とご助言を賜りました西谷滋人教授に心より深く感謝いたします．教授のご指導のおかげで，システムの仕様を定め研究を行うことができました．そして，研究の進行に伴い，様々な助力や助言，知識の供給を頂きました西谷研究室の同輩や，先輩方に深く感謝いたします．本当にありがとうございました．

参考文献

[1] module PTY - https://docs.ruby-lang.org/ja/latest/class/PTY.html (accessd on 25 Jan 2021).

[2] (Ruby) PTY.spawn("bash -i") でコマンド実行してプロンプトを expect しつつ途中の出力も随時表示する - https://memo88.hatenablog.com/entry/20180218/1518907742 (accessd on 25 Jan 2021).

[3] class IO - https://docs.ruby-lang.org/ja/latest/class/IO.html (accessd on 25 Jan 2021).

[4] instance method IO#getc - https://docs.ruby-lang.org/ja/latest/method/IO/i/getc.html (accessd on 25 Jan 2021).

[5] Sageチュートリアルへようこそ - https://doc.sagemath.org/html/ja/tutorial/index.html (accessd on 25 Jan 2021).

[6] Numerical Root Finding and Optimization - https://doc.sagemath.org/html/en/reference/numerical/sage/numerical/optimize.html (accessd on 25 Jan 2021).

付 録A　Sageクラスのソースコード

```
require "pty"
require "timeout"

$sage = "/Applications/SageMath/sage"

class Sage
  def initialize
    @define_command = Array.new
    @command_list = Array.new
    @command = ""
    @res = Array.new
    @flag = true
  end

  def add_command(com)
    @command_list << com
  end

  def define_symbol(*symbol)
    symbol = symbol.join(",")
    @define_command << "#{symbol}=var('#{symbol}')"
  end

  def exec_command
    PTY.getpty($sage) do |i, o|
      begin
Timeout.timeout(0.7) do
  loop { i.getc }
end
      rescue Timeout::Error
      end
      if @define_command != []
@command_list.unshift(@define_command)
      end
      p @command = @command_list.join(";")
      o.puts @command
      time = 5
      begin
Timeout.timeout(time) do
  loop do
    @res << i.gets
  end
end
      rescue Timeout::Error
      end
      @command_list = []
      if @flag
@res[-1].split("\e[")[-1].match(/0m(.*)\r\n/)
return $1
      else
return 0
      end
    end
  end
end
```

付 録B　Algebraクラスのソースコード

```
require "./SageMath.rb"

class Algebra < Sage
  def add(a, b)
    @command_list << "#{a}+#{b}"
  end

  def sub(a, b)
    @command_list << "#{a}-#{b}"
  end

  def mult(a, b)
    @command_list << "#{a}*#{b}"
  end

  def div(a, b)
    @command_list << "#{a}/#{b}"
  end

  def rem(a, b)
    @command_list << "#{a}%#{b}"
  end

  def factor(f)
    @command_list << "factor(#{f})"
  end

  def expand(f)
    @command_list << "expand(#{f})"
  end

  def solve(f, target)
    f.gsub!("=", "==")
    @command_list << "solve([#{f}],#{target})"
  end

  def diff(f, target, num = 1)
    @command_list << "diff(#{f},#{target},#{num})"
  end

  def integral(f, target)
    @command_list << "integral(#{f},#{target})"
  end

  def indefinite_integral(f, target, from, to)
    @command_list << "integral(#{f},#{target},#{from},#{to})"
  end

  def partial_fraction(f, x)
    @command_list << "f=#{f}"
    @command_list << "f.partial_fraction(#{x})"
  end
end
```

付 録C　Plotクラスのソースコード

```
require "./SageMath.rb"

class Plot < Sage
  def save_plot(function: "plot", plotname: "plot", extension: "png")
    @command_list << "#{function}.save('#{plotname}.#{extension}')"
    @flag = false
  end

  def set_plot(function, xmin: -5, xmax: 5)
    @command_list << "plot=plot(#{function}, (x,#{xmin},#{xmax}))"
  end

  def set_point_plot(array, color: "red", size: 30)
    @command_list << "point=list_plot(#{array},color='#{color}',pointsize=#{size})"
  end

  def fit(array, xmin: -1, xmax: 1, plotname: "plot",
 point_color: "red", curve_color: "blue")
    set_point_plot(array, color: point_color)
    define_symbol("w0", "w1", "w2", "w3")
    add_command("model(x) = w0 + w1*x + w2*x^2 + w3*x^3")
    add_command("data=#{array}")
    add_command("fit = find_fit(data, model, solution_dict=True)")
    add_command("f_fit(x) = model.subs(fit)")
    add_command("fit_plot = plot(f_fit, [x, #{xmin}, #{xmax}],
     rgbcolor='#{curve_color}')")
    add_command("p=fit_plot+point")
    save_plot(function: "p", plotname: plotname)
  end

  def animate_plot(type)
    add_command("array=[plot(c*#{type}(x), (-2*pi,2*pi), color=Color(c,0,0),
     ymin=-1, ymax=1) for c in sxrange(0,1,.2)]")
    add_command("a = animate(array)")
    add_command("a.save('plot.gif')")
  end
end
```