卒業論文

Ruby から Maple を呼び出すインターフェースライブラリ開発

関西学院大学　理工学部　情報科学科

3528 村瀬　愛理

2017 年 3 月

指導教員　　西谷　滋人　教授

目次

1　第一章 序論

1.1　Maple とは

Maple は，1980 年にカナダ・ウォータールー大学で生まれた数式処理技術をコアテクノロジーとして持つ科学・技術・工学・数学（STEM : Science, Technology, Engineering and Mathematics）に関する統合的計算環境である [Maple]. 特徴として，たくさんの数学関数が用意されていること，大きな桁数の計算が可能であること，グラフの描画が簡単であり，かつ 3 次元のグラフの描画にも対応していることなどが挙げられる．数式を入力するだけで簡単に解を得ることができることから，多くの場で用いられている．

1.2　Ruby とは

Ruby は，まつもとゆきひろ氏によって開発されたオブジェクト指向スクリプト言語である．他にもテキスト処理に適した正規表現や高階関数，ガベージ・コレクションなどの特徴を持っている．

1.2.1　Ruby と Python

Python は，Ruby と同じオブジェクト指向のスプリクト言語である．この 2 つは度々比較され，どちらが優れているのかを議論されてきた．2 つのスプリクト言語の決定的な違いは何を得意としているかである．Ruby は Web 分野を得意とするのに対し，Python は数値計算やビッグデータを得意としている．逆に Ruby は数値計算には弱く，Python は Web 分野には弱い．もちろんこの議論に答えはなく，本来は自分が何を目的としたプログラムを作るのかで使い分けるのが理想だろう．しかし，Ruby を使い慣れている人が数値計算をするためだけに Python を勉強し直すよりも，Ruby 上で数値計算ができる方が良いのは明らかである．

1.3　開発の背景

先述の通り，Ruby は数値計算関連の環境整備が遅れており，Ruby 上で高等な関数，例えば，大きな素数を求めたり，最小公倍数を求めるなどの処理を行うのが難しい．一方で，Ruby 以外の数式処理ソフトウェアなどを立ち上げて，別々に作業したり，慣れない別の言語を勉強し直したりするよりも Ruby のみでプログラミングする方が，開発速度の

格段の向上が期待できる．そこで本研究では，数式処理ソフトウェアの 1 つである Maple を Ruby 上で呼び出し，Maple に計算をさせて，その結果を Ruby が取得するインターフェースライブラリの開発を目的とする．

2 第二章 手法

2.1 Maple との通信手法

Maple は一般的に，グラフや数式の綺麗な出力や，数式の入力を初心者が直感的におこなえるように Java で作られた GUI を使って実行する．それとは別に command line で実行される計算エンジン部が用意されている．そこで, 開発する Ruby ライブラリでは，このエンジンに直接働きかけて操作する．Ruby で外部コマンドを実行する gem library の systemu を使って，出力を得るようにしている．Ruby code で要求コードを受け取った場合，そのコードを tmp.mw に書き込む．それを Maple で実行し，結果をテキストファイルで受けとることで出力を得る．

2.2 Maple 関数の類型化

今回，数多く存在する Maple の数学関数の中から整数論と行列に関するものを選抜し実装した．図 1 実装した整数論に関する関数の役割と入出力図 2 実装した行列に関する関数の役割と入出力

2.3 出力の切り替え

Maple から受け取ったままの出力は，値の前にスペースがたくさん入っていることや，出力が String 型であることから，その数値を使って計算をするようにプログラミングしていた場合に支障をきたす．このため，関数ごとに正しい型で出力できるように wrapper を作る．例えば，int 型で出力が欲しいものは exec を exec_i から呼び出すことで対応する．このように boolean や float といった出力型に応じて，exec_b，exec_f のように関数を増やしていく．

2.3.1 出力の切り替えの例 - 整数論の場合

まず，nextprime を例にとると

```
require "mapleruby/version"
require 'systemu'
require 'yaml'
```

```
class RMaple
#整数論
def nextprime(a)
a = a.to_i
p Mapleruby.new("nextprime(#{a})").exec_i
end
end
class Mapleruby
DATA_FILE=File.join(ENV['HOME'],'.mapleruby_rc')
# Your code goes here...
def initialize(maple_code)
@maple_code = maple_code
@src = get_env
@maple_path=@src[:MAPLE_PATH]
end
def exec_i
result = exec
return result.to_i
end
def exec
 code0=<<EOS
interface(quiet=true);
writeto("./result.txt");
#{@maple_code};
writeto(terminal);
interface(quiet=false);
EOS
 File.write('tmp.mw',code0)
 command="#{@maple_path} tmp.mw"
 status,stdout,stderr=systemu command
 status,stdout,stderr=systemu 'cat result.txt'
# result=stdout
  # print(result)
```

```
 return stdout
end
def get_env
begin
file = File.open(DATA_FILE,'r')
rescue => evar
p evar
print "no resource file for mapleruby.\n"
end
@src = YAML.load(file.read)
file.close
p @src
end

end
```

このように exec_i は，exec を通った値を to_i し，int 型にしてから返すようになっている．もし使われた関数が isprime だった場合は exex_b から返されるときに，boolean 型の true と false が返されるようになっている．

2.3.2 出力の切り替えの例 - 行列