

Ruby-Gnuplot を用いた数値計算の視覚化 を効率良く行う手法の開発

情報科学科 西谷研究室 4697 西坂謙一

平成 20 年 2 月 21 日

概要

数値計算とは、解析的に解くことが不可能な数学の問題において、計算機科学を駆使して数値的に求める手法に関する学問である。あらゆる研究において、シミュレーションにより現象を数値的かつ視覚的に理解することは、様々な発見につながる非常に有効な手段である。数値計算においても同様で、それは従来、Maple などの数式処理システムによって実現されてきた。

しかし、現在西谷研究室で研究を進めている、関学の金子教授らにより開発された、SiC の液相成長を実現する準安定溶媒エピタキシー (Metastable Solvent Epitaxy) と呼ばれる新プロセスなどのシミュレーションプログラムを、Maple 独自の開発環境や言語で書くことは非常に効率が悪いと言える。このようなシミュレーションを行う際、互換性や自由度、実行速度などを備えた言語の方が、開発段階で効率が良い。

そこで本研究では、Maple よりも他のソフトとの連携が容易で、エディターなどのプログラミング環境が整った言語、ruby でのスクリプトから、直接視覚化を実現することを目的とした。手法として、プログラムから得られた数式やデータを、描画プログラムへ直接送り、視覚化を効率良く実現する機構を構築する。実現方法として、視覚化を実現するライブラリーを作成した。

Maple を参考に、シミュレーション実現での利用頻度の高い機能を選定し、ライブラリーを作成、その用途と利用方法を記した。また、実行例として、準安定溶媒エピタキシー (Metastable Solvent Epitaxy) によるシミュレーションを本ライブラリーを用いて視覚化し、その結果を報告する。また、本ライブラリーのひな形にあたる部分を第 4 章に載せている。

目次

第1章	緒言	4
第2章	実現方法	7
2.1	実現手段の選定	7
2.1.1	Ruby	7
2.1.2	Gnuplot	8
2.1.3	AquaTerm	8
2.2	実現方法	9
2.3	視覚化機能の比較	10
第3章	使用方法	12
3.1	plot	12
3.1.1	plot	12
3.1.2	plots	13
3.1.3	plot3d	14
3.2	animation	15
3.2.1	animate	15
3.2.2	listplot	17
3.2.3	listplots	18
3.2.4	listplot3d	20
第4章	プログラム解説	22
4.1	ruby による実行ファイル	22
4.2	ライブラリーのひな形	24
第5章	結果	36
第6章	総括	38

付 録 A	41
A.1 MSE2D 計算プログラム	41
A.1.1 mse2d.rb	41
A.1.2 autocalc-mse2d.rb	48
A.2 Mandelbrot 集合計算プログラム	50
A.3 QuickTime ファイルへの書き出し	51

第1章 緒言

実際に実験を行うことが困難であったり，現実的に不可能である場合，シミュレーションにより検証を行うことは有効な手段である．数値計算に使用される Maple とは，専用のプログラミング言語を用いる数式処理，数式モデル設計パッケージである．Maple では足し算から微分方程式求解やテイラー展開，プロットやアニメーションまでを網羅する機構を実現することで，利用者の計算環境を整えることを目的にしており，現象を解析したり理解したい学生や研究者にとって，非常に有用なソフトウェアである．

数式処理ソフト Maple では，Maple スクリプトと呼ばれる独自のプログラミング言語が用意されている．しかし，その構文は独自の仕様であり，さらに基本的に Maple ソフト中でコーディング，実行，デバックを行う必要がある．これではプログラム作成，実行が効率的に行うことができない．一方，ruby はもっとも遅く開発されたオブジェクト指向スクリプト言語であり，構文が非常に洗練されている．さらに，Emacs などのエディターを使って効率的にコーディング開発，デバックする環境が提供されている．また，スクリプト言語の特徴として，外部呼び出しが容易であったり，文字列操作機能にも優れており，データファイルに書かれたデータを読み書きすることが容易である，というメリットが挙げられる．

スクリプト上での数値計算の視覚化の実現には，他にも既存の数値計算のライブラリーを利用する方法が挙げられる．代表的な数値計算視覚化ライブラリーである RubyDCL を用いて，視覚化を実行する ruby コマンドの例を以下に記す．RubyDCL.rb は，等高線を塗り分ける視覚化を実現する RubyDCL を，呼び出すコマンドの例である．RubyDCL を利用する場合と同様に，一般的にライブラリーを使用した場合，コマンドが複雑であり，スクリプト自体も冗長なものとなる．初学者にとって，スムーズに視覚化を実現することは困難である．

そこで本研究では，プログラムから得られた数式やデータを，グラフ作成アプリケーションソフトウェアである gnuplot へ直接送る機構を構築

する．それにより，プログラムによって得られたデータを，スムーズに視覚化することを可能とする．実現方法として，多数ある既存の数値計算ライブラリーを使用するのではなく，`plot` や `animation` などのシミュレーション機能に特化した，新たなライブラリーのひな形を `ruby` によってコーディングする．また，それらを使用する実行ファイルも作成する．本ライブラリーを使用することによって，ユーザーに使いやすい形で Maple 並の視覚化機能を実現することを目的とした．本ライブラリー作成にあたっては，視覚化の対象としてシミュレーション実現での重要度が高い `plot`, `plots`, `plot3d`, `animation`, `listplot`, `listplot3d` をひな形として選定した．

```

DCL::grswnd(tmin, tmax, zmin, zmax)          \# 左・右・下・上
の座標値
DCL::uspfrit                                \# その他は適
当に決める
DCL::grstrf                                  \# 座標の確定

DCL::sglset('lfprop',true)                  \# プロポーシ
ヨナルフォント
DCL::ussttl('TIME', 'YEAR', 'HEIGHT', 'km') \# タイトルの設
定
DCL::uelset("ltone",true) if ARGV.index("-color") \# 色つ
き
DCL::uetone(u)                               if ARGV.index("-color") \# めり
わけ
DCL::usdaxs                                  \# 座標軸を描
く
DCL::udcntr(u)                               \# 等値線をひ
く

DCL::sgstxs(0.02)                            \# 文字の大き
さの設定
DCL::sgtxr(0.7,0.9, "umin="+u.min.to\_s+" umax="+u.max.to\_s)
\# to\_s 文字
列に変換
require "date"
DCL::sgtxr(0.75,0.85, "created by \#{\$0} \#{Date.today}")

DCL::grcls

```

第2章 実現方法

2.1 実現手段の選定

2.1.1 Ruby

一般的に、ruby は数値計算には向かないと言われる。実際に、ruby は C や Fortran などの他の言語と比べ、実行速度などの面で遅い。しかし、開発環境や他ソフトとの連携を考慮すると、ruby により数値計算プログラムを書くことは有効であると考えられる。数値計算をする場合、一般には入力があって出力がある。両者が少数の数値のみからなる場合はほとんど問題はないが、現実的な問題では、入力も出力も単純、少量でなくなる。さらに、似たようなことをするのに複数の形式のデータを扱うこともある。このような場合に、オブジェクト指向言語である ruby は、クラスを導入することでプログラムの見通しが良くなり、可読性・保守性の向上で、開発が容易になる。

また、ruby は C で実装されており、C で容易に拡張ライブラリーを書くことができる。ruby 組み込みのクラスの多くは C で書かれており、それを ruby で利用できるようにする仕組みがそのまま使えるので、容易であるだけでなく、移植性も高い。C だけでなく、これまでに多くの数値計算ライブラリーが開発されてきた言語 FORTRAN77 のライブラリーを利用することも、比較的容易である。C や FORTRAN の既存のライブラリーを ruby に組み込むことで、より使いやすい形に拡張できる。これらの理由から本研究では、計算速度自体よりも、開発の容易さや拡張性、可読性、保守性などの点でライブラリーのコーディングに ruby を選定した。

2.1.2 Gnuplot

gnuplot はコマンド駆動型の対話型関数描画プログラムである。現在、Linux, UNIX, Windows, Mac OS などの多くの OS に対応したバージョンが開発されている。画像ファイルも様々な書式で保存が可能である。高機能であることから、広く利用され、オンラインマニュアルも多数存在している。ところが、gnuplot ではユーザーインターフェースにおいて、ユーザーがコマンドを入力する形式になっている為、コマンドを理解して覚えておく必要がある。

しかし、様々なシミュレーション、プロットにおいて gnuplot による 2 次元、3 次元グラフ描画機能でカバーできないことは稀であり、その描画機能でユーザーの意図した視覚化が可能である。さらに、gnuplot はスクリプトにも適している。対話型実行中に追加コマンドを含むファイルを読み込むことができ、既に存在するファイルや標準入力からのコマンド列をパイプを使ってバッチモードでそれを処理することもできる。よって、ruby からのコントロールファイルの受け取りが可能で、高描画機能を持つ、gnuplot を採用した。

2.1.3 AquaTerm

グラフィックターミナルソフトは AquaTerm を使った。AquaTerm とは Mac OS X の Aqua ウィンドウ環境にベクトルグラフィックスを表示するフロントエンドアプリケーションである。gnuplot 等のアプリケーションから、API を介して描画要求を受け取り、グラフをウィンドウ,PDF,EPS に出力する。無償であり、基本的な描画はこれ一つで十分であった。

2.2 実現方法

数値計算とその視覚化の実現を，ruby によってコーディングされたライブラリーと gnuplot によって行う．具体的には，ruby では視覚化が不可能な為，gnuplot へ実行ファイルを吐き出す為の実行ファイルとライブラリーを作成する．

gnuplot では本来，ユーザーにより数式やデータのほかに変域や値域，プロットする範囲などを gnuplot 独自のコマンドで入力する必要があり，一般ユーザーには使いにくいものとなっている．また，高度なアニメーション表示などになるとコントロールファイルを作成するのにかなりの時間を要する．そこで本ライブラリーでは，構文解析を実行することにより，ユーザーは変域や値域，プロット範囲を羅列するだけで自動的に gnuplot が読み込める形式のファイルを作り出す機構を構築する．また，リストプロットなどのアニメーション表示を行う場合には，数式かデータファイルを指定するだけで Quicktime ファイルへ書き出すコントロールファイルを作り出すようにした．

さらに，機能の面では，maple を参考に option などの機能も追加している．その他，全体をライブラリー化することで，本体を極めて簡潔なものとし，可読性の向上，高度な視覚化に対応する為の書き換え，追記による改良に備えた．

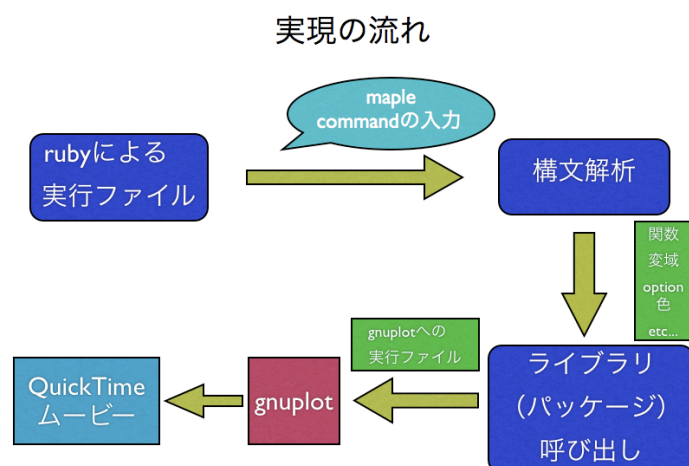


図 2.1: 実現方法イメージ

2.3 視覚化機能の比較

ここで，実現するにあたって重要となる，視覚化機能の比較を見る．
gnuplot と maple によるそれぞれの視覚化機能について比較させた．以下のグラフは同じ式，同じ初期値を与えたそれぞれのシミュレーション結果の一部を示している．

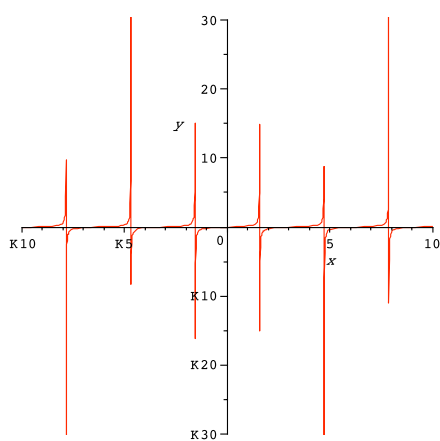


図 2.2: maple による $\tan(x)$ プロット

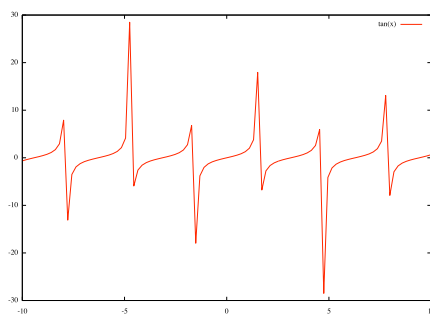


図 2.3: gnuplot による $\tan(x)$ プロット

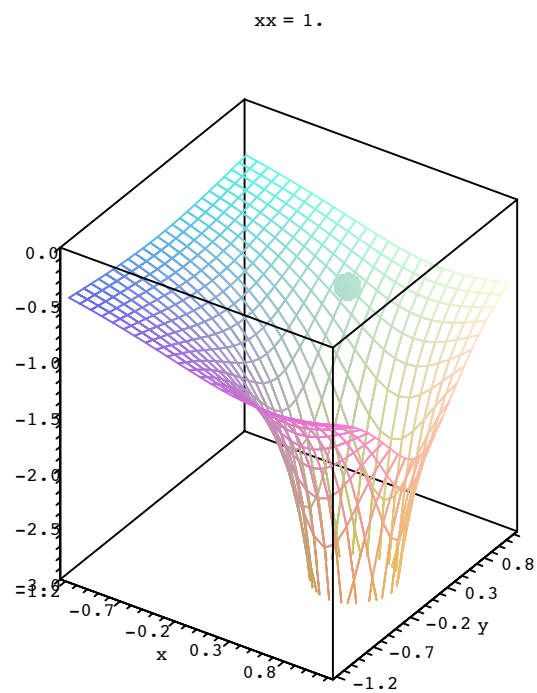


図 2.4: maple による重力場シミュレーション

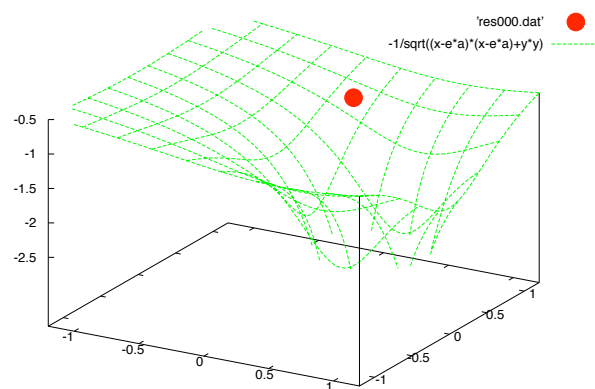


図 2.5: gnuplot による重力場シミュレーション

第3章 使用方法

この章では、本システムの用途と用途別の具体的な使い方について解説する。

3.1 plot

3.1.1 plot

最も基本的なグラフの視覚化であり、

```
command >> plot f(x) x[x.min..x.max] y[y.min..y.max] option = ...
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> plot sin(x) x[-10..10] y[-10..10] option = linespoints
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる。

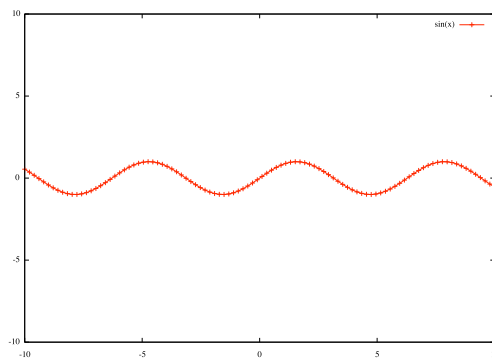


図 3.1: $\sin(x)$ の linespoint を用いたグラフィック出力

3.1.2 plots

複数のグラフの plot に用いる .

```
command >> plot f(x), g(x), .. x[x.min..x.max] y[y.min..y.max] option =  
...
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> plots sin(x), cos(x), tan(x) x[-10..10] y[-10..10] option =  
points
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる .

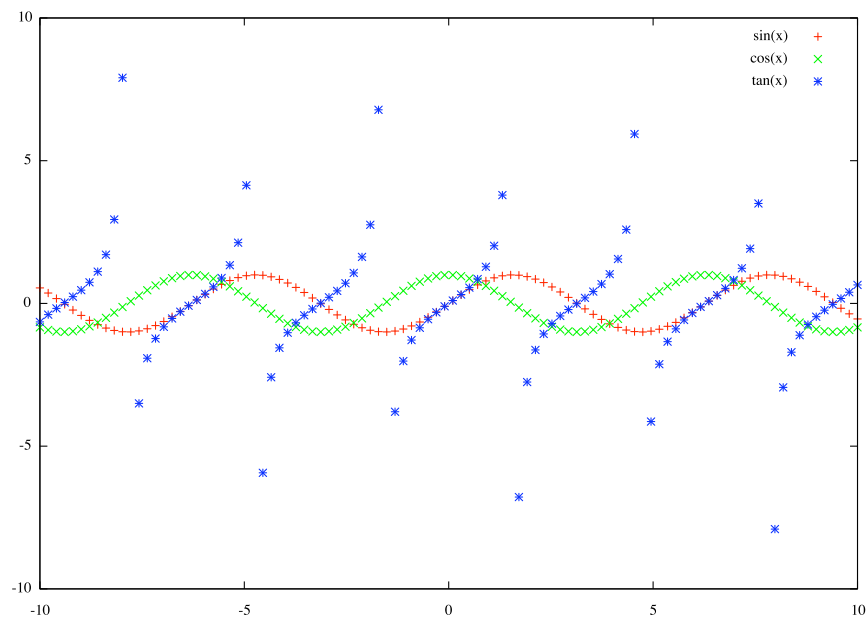


図 3.2: $\sin(x)$, $\cos(x)$, $\tan(x)$ の points を用いたグラフィック出力

3.1.3 plot3d

グラフの3次元表示に用いる．複数の同時plotにも対応している．

```
command >> plot3d f(x), g(x), .. x[x.min..x.max] y[y.min..y.max] z[z.min..z.max]  
option = ...
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> plot3d sin(x*x), cos(x*x), tan(x*x) x[-2..2] y[-2..2] z[-1..1]
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる．

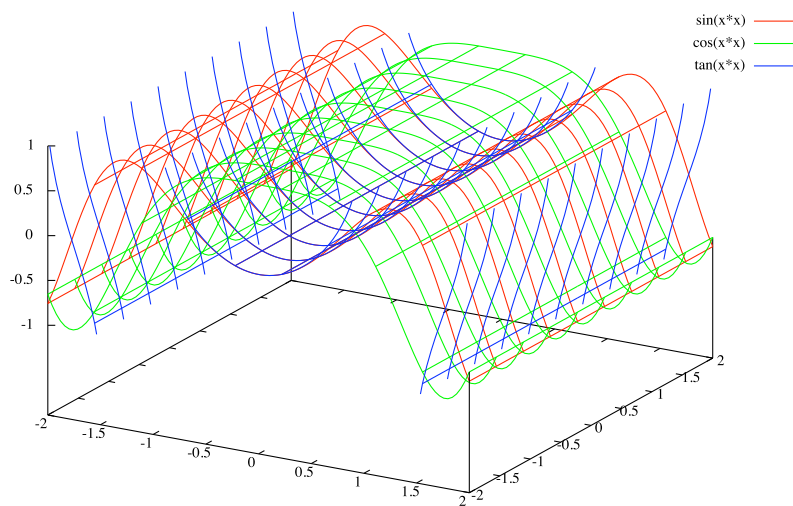


図 3.3: $\sin(x^2), \cos(x^2), \tan(x^2)$ の3次元グラフィック出力

3.2 animation

3.2.1 animate

基本的なグラフのシミュレーションに対応する `.gnuplot` によるアニメーションは複数のイメージを次々と見せていく, gif アニメと同じ仕様となっている. 複数枚のイメージを作成し, それを任意の秒数おきに表示するコントロールファイルも同時に作成する.

```
command >> animate f(x), g(x), .. x[x.min..x.max] y[y.min..y.max] option =  
...
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> animate sin(x), cos(x) x[-10..10] y[-1.5..1.5]
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる.

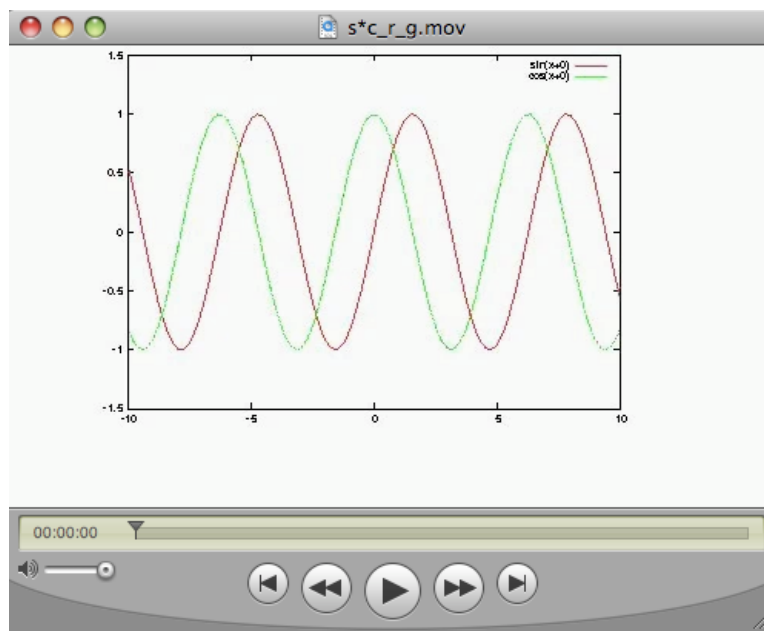


図 3.4: $\sin(x), \cos(x)$ の時間変化シミュレーション

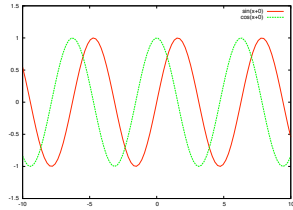


図 3.5: $\sin(x), \cos(x)$ の時間変化
シミュレーション初期状態

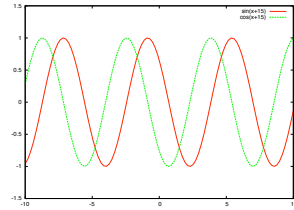


図 3.6: $\sin(x), \cos(x)$ の時間変化
シミュレーション 15 枚目

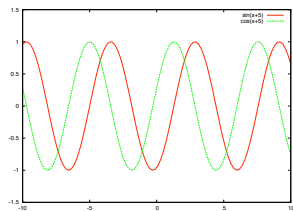


図 3.7: $\sin(x), \cos(x)$ の時間変化
シミュレーション 5 枚目

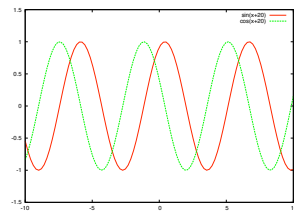


図 3.8: $\sin(x), \cos(x)$ の時間変化
シミュレーション 20 枚目

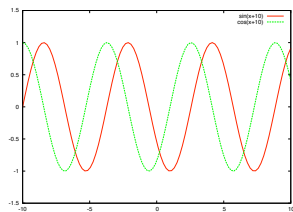


図 3.9: $\sin(x), \cos(x)$ の時間変化
シミュレーション 10 枚目

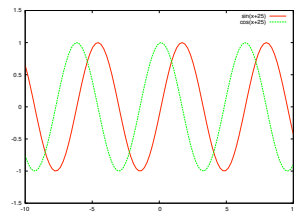


図 3.10: $\sin(x), \cos(x)$ の時間変化
シミュレーション 25 枚目

3.2.2 listplot

listplot は同一ファイルの複数データに対応している .

```
command >> listplot filename x[x.min..x.max] y[y.min..y.max]
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> listplot Mandelbrot.dat
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる .

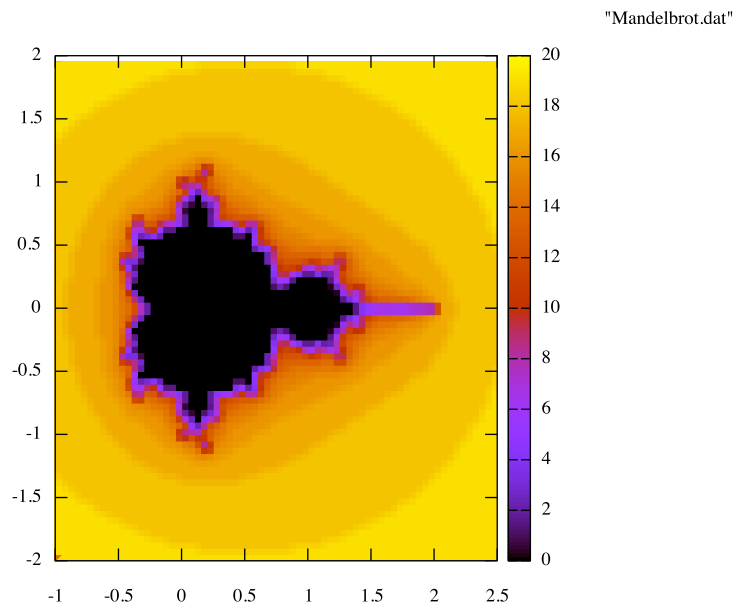


図 3.11: Mandelbrot 集合の描画

3.2.3 listplots

listplots は複数枚の複数のデータのシミュレーションに対応している .
command >> *listplots filename x[x.min...x.max] y[y.min...y.max]* データ数

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> listplots res x[0..30] y[0..300]100
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる .

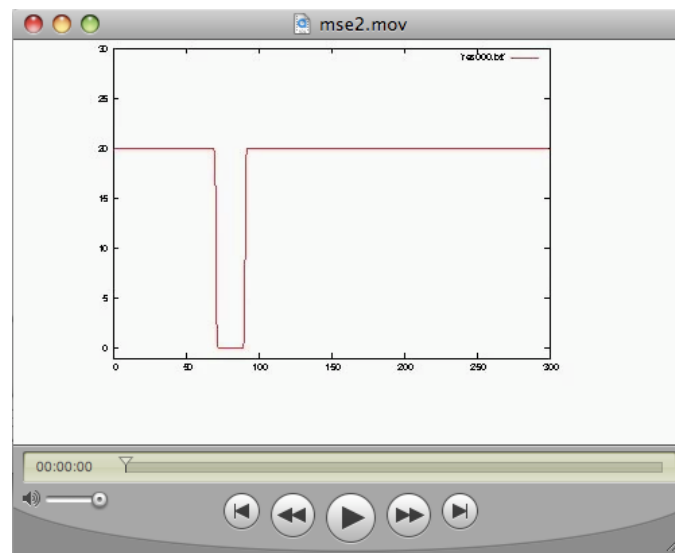


図 3.12: MSE 状態変化シミュレーション

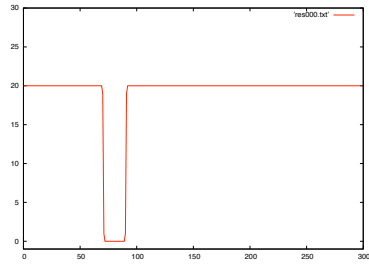


図 3.13: mse 状態変化シミュレーション初期状態

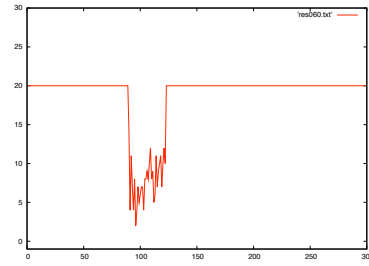


図 3.14: mse 状態変化シミュレーション 60 枚目

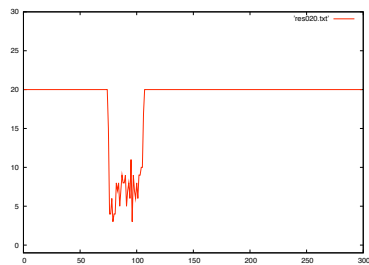


図 3.15: mse 状態変化シミュレーション 20 枚目

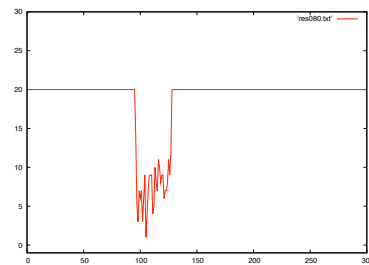


図 3.16: mse 状態変化シミュレーション 80 枚目

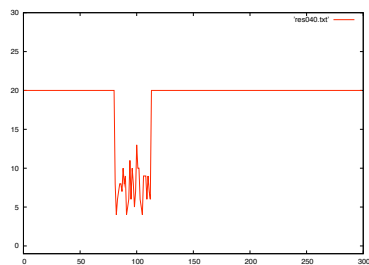


図 3.17: mse 状態変化シミュレーション 30 枚目

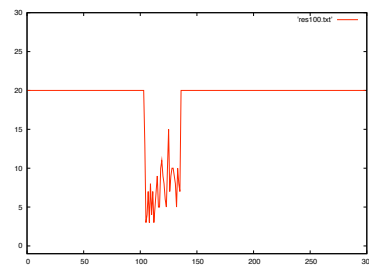


図 3.18: mse 状態変化シミュレーション 100 枚目

3.2.4 listplot3d

listplots と同様，複数枚のデータを 3D でアニメーション表示する．

```
command >> listplot3d filename x[x.min..x.max] y[y.min..y.max] z[z.min..z.max]
```

コマンド例は

```
user%ruby ruby - gnuplot.rb
```

```
command >> listplot3d res x[-1..1] y[-1..1] 39
```

```
output >>> orange.gpl
```

```
user%gnuplot orange.gpl
```

となる．

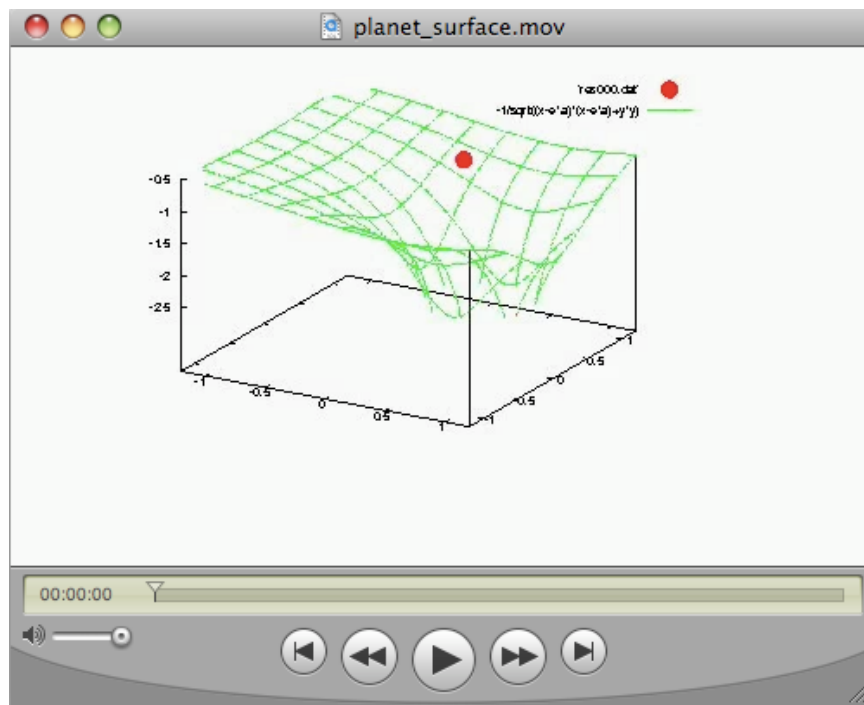


図 3.19: 重力場シミュレーション

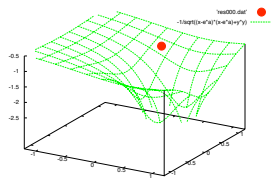


図 3.20: 重力場シミュレーション
初期状態

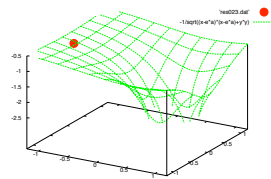


図 3.21: 重力場シミュレーション
60 枚目

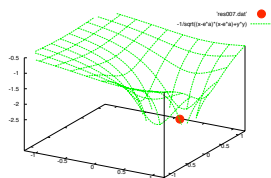


図 3.22: 重力場シミュレーション
20 枚目

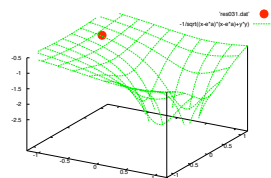


図 3.23: 重力場シミュレーション
80 枚目

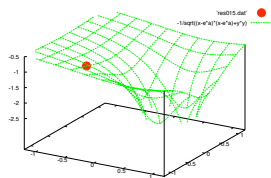


図 3.24: 重力場シミュレーション
40 枚目

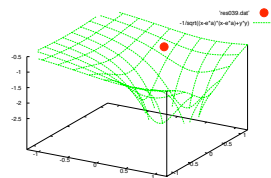


図 3.25: 重力場シミュレーション
100 枚目

第4章 プログラム解説

4.1 ruby による実行ファイル

全体をライブラリー化し、本体は極めて簡潔なものとしている。

```
include Math
require 'pp'

//orange library の呼び出し
Dir.chdir("lib_orange"){
  require 'plot_con'
  require 'plots_con'
  require 'plot3d_con'
  require 'anime_con'
  require 'listplot_con'
  require 'listplots_con'
  require 'listplot3d_con'
  require 'option'
}

print 'maple command >>>'
temp = gets.chomp

com = temp.split(" ")

//command の内容により gnuplot へ吐くファイルの contents 部分
を返り値とする .
if(com[0]=='plot')
```

```

contents = plot_con(com)

elsif(com[0]=='plots')
contents = plots_con(com)

elsif(com[0]=='plot3d')
contents = plot3d_con(com)

elsif(com[0]=='animate')
contents = anime_con(com)

elsif(com[0]=='listplot')
contents = listplot_con(com)

elsif(com[0]=='listplots')
contents = listplots_con(com)

elsif(com[0]=='listplot3d')
contents = listplot3d_con(com)
else
puts 'Errorr'
end

```

//ライブラリーによって，contents 部分には plot 範囲や plot の種類，animation の場合にはコントロール部分が書き込まれている．

```

filename = 'orange.gpl'
File.open filename,'w' do |f|
  f.write contents
end

```

```
puts 'output >>> orange.gpl'
```

//orange.gpl を gnuplot で実行することで，目的のグラフィックが出力される．

4.2 ライブラリーのひな形

plot_con.rb

```
def plot_con(com)

  b = com[2].split("(")
  c = b[1].split("..")
  d = c[1].split(",")
  e = com[3].split("(")
  f = e[1].split("..")
  g = f[1].split(",")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s

  contents = 'set xrange [' + x_min + ':' + x_max + ']' +
  "\n" +
  'set yrange [' + y_min + ':' + y_max + ']' +
  "\n" +
  'plot ' + com[1].to_s + opt(com) + "\n"

  return contents
end
//構文解析によって得られた，入力したf(x)を，
//プロットをする実行ファイルを返す．
```

```

def plots_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s

  plots = ' '
  h = com[1].split(",")
  length = h.size

  for i in 0..length-1
    if(i<length-1)
      plots = plots + h[i].to_s + opt(com) + ","
    else
      plots = plots + h[i].to_s + opt(com)
    end
  end
  contents = 'set multiplot' + "\n" +
    'set xrange [' + x_min + ':' + x_max + ']' +
    "\n" +
    'set yrange [' + y_min + ':' + y_max + ']' +
    "\n" +
    'plot ' + plots + "\n"

  end
return contents
end
//構文解析によって得られた, 入力した f(x), g(x).. を,
プロットする実行ファイルを返す.

```

```
def plot3d_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")
  h = com[4].split("[")
  i = h[1].split("..")
  j = i[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s
  z_min = i[0].to_s
  z_max = j[0].to_s

  splot = ' '
  k = com[1].split(",")
  length = k.size
  for i in 0..length-1
    if(i<length-1)
      splot = splot + k[i].to_s + opt(com) + ","
    else
      splot = splot + k[i].to_s + opt(com)
    end
  end
  //構文解析によって得られた, 入力した f(x) を,
  3次元プロットする実行ファイルを返す.
```

```
contents = 'set multiplot' + "\n" +  
           'set xrange [' + x_min + ':' + x_max + ']' +  
           "\n" +  
           'set yrange [' + y_min + ':' + y_max + ']' +  
           "\n" +  
           'set zrange [' + z_min + ':' + z_max + ']' +  
           "\n" +  
           'splot '+ splot + "\n"  
  
end  
return contents  
end
```

//構文解析によって得られた，入力した $f(x), g(x), \dots$ を，
3次元プロットする実行ファイルを返す．

```

def anime_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s

  anime = ''
  h = com[1].split(",")

  fx = []
  h.size.times do |i|
    fx.push h[i].split("(x)") [0]
  end

  for j in 0..8*PI

    anime=anime+'plot '
    fx.size.times do |i|
      if(i<fx.size-1)
        anime=anime+fx[i]+'(x'+j.to_s+')'+opt(com)+'',
      else
        anime=anime+fx[i]+'(x'+j.to_s+')'+opt(com)+"\n"
      end
    end
    anime += 'pause 1'+"\n"
  end
end

```

— anime_con.rb —

```
contents = 'set xrange [' + x_min + ':' + x_max + ']' +
  + "\n" +
  'set yrange [' + y_min + ':' + y_max + ']' +
  + "\n" +
  anime

return contents
end
```

//構文解析によって得られた, 入力した $f(x), g(x), \dots$ を,
アニメーションにする実行ファイルを返す.

```
def listplot_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s

  contents = 'set xrange [' + x_min + ':' + x_max + ']'
    + "\n" +
    'set yrange [' + y_min + ':' + y_max + ']'
    + "\n" +
    'plot ' + '\',' + com[1] + '\',' + opt(com) + "\n"
  return contents
end
```

//構文解析によって得られた，入力したファイルに格納されたデータを，プロットする実行ファイルを返す．

```

require 'pp'

def listplots_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s

  plot = []
  out = []
  com[4] = com[4].to_i
  for i in 0..com[4]
    if(i<10)
      number = '00'+i.to_s
    elsif(i>=10 && i<100)
      number = '0' + i.to_s
    else
      number = i.to_s
    end
    a = 'plot \'' + com[1] + number + '.txt\'' + opt(com)
    + "\n" + 'pause 1' + "\n" + "\n"
  end
end

```



```

    plot.push a
    out.push 'set output \'' + com[1] + number + '.eps\''
    + "\n"+a
end

ter = 'set terminal postscript enhanced color eps'
+ "\n"

range = 'set xrange [' + x_min + ':' + x_max + ']'
    + "\n" +
    'set yrange [' + y_min + ':' + y_max + ']' + "\n"

contents = range
ter = ter + range
for i in 0..plot.size-1 do
    contents = contents + plot[i]
    ter = ter + out[i]
end
//構文解析によって得られた，入力したファイルに格納された
データを，アニメーションにする実行ファイルを返す．

filename2 = 'save_eps.gpl'
File.open filename2, 'w' do |f|
    f.write ter
end
puts 'made save_eps.gpl'
return contents
end
//QuickTime ファイルへ書き出す為のイメージを，
eps ファイルとして保存するファイルを同時に作成する．

```

```

def listplot3d_con(com)
  b = com[2].split("[")
  c = b[1].split("..")
  d = c[1].split("]")
  e = com[3].split("[")
  f = e[1].split("..")
  g = f[1].split("]")
  h = com[4].split("[")
  i = h[1].split("..")
  j = i[1].split("]")

  x_min = c[0].to_s
  x_max = d[0].to_s
  y_min = f[0].to_s
  y_max = g[0].to_s
  z_min = i[0].to_s
  z_max = j[0].to_s

  contents = 'set pm3d' + "\n" +
    'set xrange [' + x_min + ':' + x_max + ']' +
    + "\n" +
    'set yrange [' + y_min + ':' + y_max + ']' +
    + "\n" +
    'set zrange [' + z_min + ':' + z_max + ']' +
    + "\n" +
    'splot ' + '\',' + com[1] + '\',' + opt(com) + "\n"
  return contents
end

```

//構文解析によって得られた，入力したファイルに格納されたデータを，3次元アニメーションにする実行ファイルを返す．

```
def opt(com)
  l = com.include?("option=lines")
  p = com.include?("option=points")
  lp = com.include?("option=linespoints")
  d = com.include?("option=dots")
  i = com.include?("option=impulses")

  if(l==true)
    flag = 'lines'
  end
  if(p==true)
    flag = 'points'
  end
  if(lp==true)
    flag = 'linespoints'
  end
  if(d==true)
    flag = 'dots'
  end
  if(i==true)
    flag = 'impulses'
  end

  if(flag=='lines')
    temp = ' with lines'
  elsif(flag=='points')
    temp = ' with points'
  elsif(flag=='dots')
```

```
temp = ' with points'
elsif(flag=='dots')
temp = ' with dots'
elsif(flag=='linespoints')
temp = ' with linespoints'
elsif(flag=='impulses')
temp = ' with impulses'
else
temp = ''
end
return temp
end
```

//返り値に option のコマンドを追加する .

第5章 結果

作成したライブラリーにより，現在研究を進めている準安定溶媒エピタキシー (Metastable Solvent Epitaxy) によるシミュレーションを2次元で実際に視覚化し，以下のように得られた．

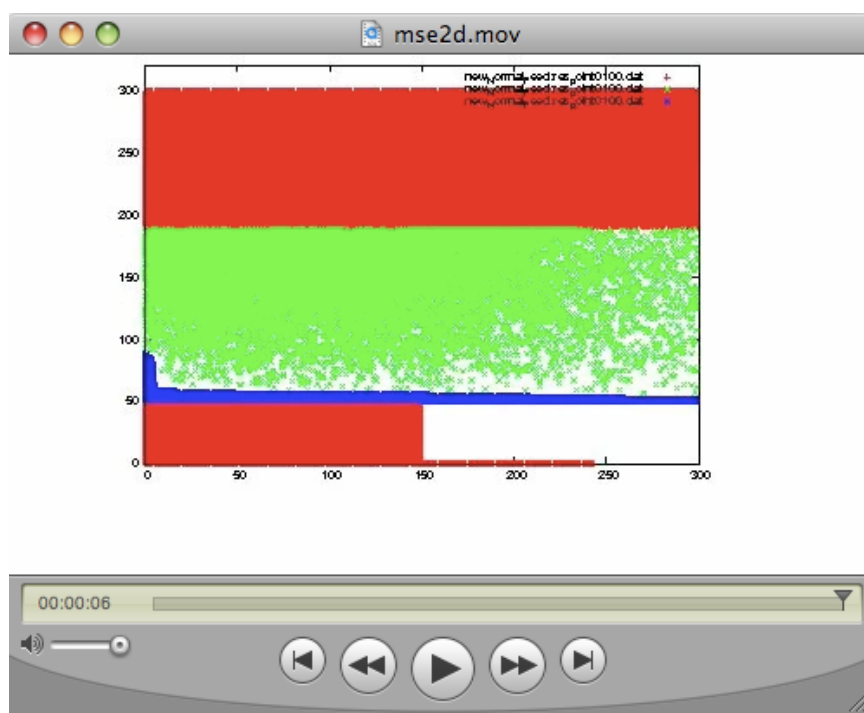


図 5.1: mse2D 状態変化シミュレーション

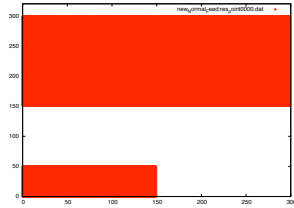


図 5.2: mse2D 状態変化シミュレーション初期状態

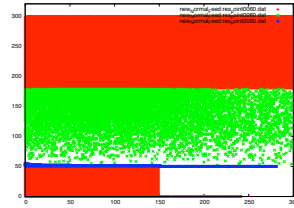


図 5.3: mse2D 状態変化シミュレーション 60 枚目

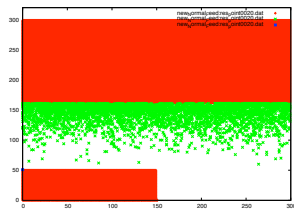


図 5.4: mse2D 状態変化シミュレーション 20 枚目

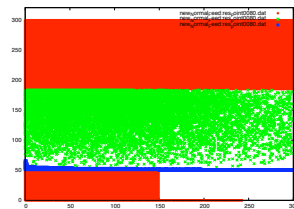


図 5.5: mse2D 状態変化シミュレーション 80 枚目

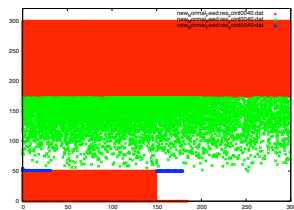


図 5.6: mse2D 状態変化シミュレーション 40 枚目

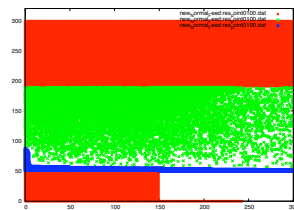


図 5.7: mse2D 状態変化シミュレーション 100 枚目

第6章 総括

本研究では，数値計算のシミュレーションにおける，グラフィック出力に焦点をあてて研究を進めてきた．本研究によって，以下のような成果が得られた．

1. gnuplot への入力ファイルを作成する，ruby ライブラリーをコーディングした．
2. Maple と同等の容易さでのスクリプトを書くことができ，同等の出力結果を実現することができた．

更なるライブラリーの拡充により，シミュレーションの実現精度を上げること，ユーザーに対するヘルプなどを作ることが今後の課題である．

謝辞

本研究を遂行するにあたり，終始多大なる有益なご指導，及び丁寧な助言を頂いた西谷滋人教授に深い感謝の意を表します．

また，本研究を進めるにつれ，西谷研究室員の皆様にも様々な知識の供給，ご協力を頂き，本研究を大成する事ができました．最後になりましたが，この場を借りて心から深く御礼申し上げます．

参考文献

- [1] 小国 力著 .『Maple V と利用の実際』
- [2] 関西学院大学，博士論文，2004 年 1 月，浅岡康著 .『極薄液層環境を用いた単結晶 SiC 薄膜成長に関する研究』
- [3] 高橋征義著 .『Ruby レシピブック』
- [4] 高橋征義，後藤裕蔵，共著 .『楽しい Ruby』
- [5] 「Gnuplot 入門 第 0.08 版」 半場 滋
< <http://dsl4.eee.u-ryukyu.ac.jp/DOCS/gnuplot/gnuplot.html> >
(2008/2/5 アクセス)
- [6] 「rubyist magazine」 日本 ruby の会
< <http://jp.rubyist.net/magazine/?0006-RLR> >
(2008/2/8 アクセス)
- [7] 「電脳 Ruby プロジェクトホームページ」 地球流体電脳倶楽部
< <http://ruby.gfd-dennou.org/tutorial/gokuraku/> >
(2008/2/10 アクセス)

付 録 A

A.1 MSE2D 計算プログラム

MSE2D のシミュレーションを行うにあたって , ポイントを計算するプログラムとその autocalc

A.1.1 mse2d.rb

```
mse2d.rb

def mse2d(title)

  t0 = Time.now
  tms0 = Process.times

  $crystal = []
  $solute = []
  $solvent = []
  solute_ave = 0
  $time2 = 0
  $flag_Attach = 0
  $rock = 0
  first_count0 = 0
  first_seed = []
  first_feed = []
  break_flag = 0
  kk_flag = 0
  $edge_count = 0
  $ledge_count = 0
```

```

ff_p = []
Dir.chdir("all_def")
require 'make_cell'
require 'checkConcentration'
require 'atomswap'
require 'feed'
require 'seed'
require 'plot'
require 'trymove'
require 'escape'
Dir.chdir("..")

#-----main

$array2 = make_cell($atom_hei,$length,$length,
$seedfront[0][1]+1,$feedfront[1]-1)

puts 'seedfront = ['+$seedfront.join(',')+']'
puts 'feedfront = ['+$feedfront.join(',')+']'
first_seed.push [$seedfront[0][0],$seedfront[0][1]]
first_feed.push [$feedfront[0],$feedfront[1]]
#print_matrix($array2,$height)
puts '0 = '+(first_count0=count0()).to_s

kk = 0
Dir.chdir(ARGV[0]) #今いる path をカレントディレクトリから
(ARGV[0]) に移す

checkConcentration_seed_point()
make_res_point(kk,$cc_point)

ff_p.push $feedfront[0]
puts 'check0'
for mse in 1..$time

```

```

if($solute.length<$select)
  feedDetach()
  #seedDetach()
  solutehead = 0
  while(solutehead<$solute.length-1)
    trymove(solutehead)
    if($flag_Attach==1)
      solutehead = solutehead-1
      $flag_Attach = 0
    end
    solutehead += 1
  end

else
  solute_array = []
  for solutehead in 0..$solute.length-1
    solute_array.push solutehead
  end
  flag = 0
  while(flag!=1)
    feedDetach()
    #seedDetach()
    $select.times do
      if(solute_array.length>0)
        n = rand(solute_array.length)
        trymove(solute_array[n])
        if($flag_Attach==0)
          solute_array.delete_at(n)
        else
          solute_array.pop
          $flag_Attach = 0
        end
      end
    end
  end
end

```

```

        else
            flag = 1
            break
        end
    end
end
end
solute_ave += $solute.length

if ((mse % 100) == 0)
    kk += 1
    checkConcentration_seed_point()
    if($crystal.length>0 && kk_flag==0)
        pp = kk
        kk_flag = 1
    end
    #make_res_atom3d(kk,$concentration)
    #make_res_seed2d(kk,$cc_seed2d)
    make_res_point(kk,$cc_point)
    puts 'check'+mse.to_s
end
$time2 = kk - 1
ff_p.push $feedfront[0]
if($feedfront[1]-$seedfront[$seedfront.length-1][1]<3 ||
$feedfront[1]==$length-1)
    puts 'break'
    break_flag=1
    break
end
end
$time2 += 1

```

```

#puts
#puts print_front()
#puts
#puts 'feedfront = ['+$feedfront.join(',')+']'
#print_matrix($array2,$height)
puts '0 = '+count0().to_s
puts 'solute_average = '+ (solute_ave/$time).to_s
puts 'ledge = '+ $ledge_count.to_s
puts ' edge = '+ $edge_count.to_s

make_anime_point(title,0,0,$height,$length+20,$time2,pp)
save_eps_point(title,0,0,$height,$length+20,$time2,pp)
make_change_eps_to_gif_point($time2)

res_f = File.open("res_feed.dat",'w')
for i in 0..ff_p.length-1
  res_f.printf("%d %d\n",i,ff_p[i])
end
res_f.close

file_f = File.open("feedfront.gpl",'w')

file_f.printf("set xrange[0:%d]\n",ff_p.length)
file_f.printf("set yrange[0:%d]\n",$height+10)
file_f.printf("plot \'res_feed.dat\' w l\n",ff_p[i])

file_f.close

t1 = Time.now
tms1 = Process.times
puts '実時間 = '+ (t1-t0).to_s
puts 'CPU時間 = '+ (tms1.utime-tms0.utime).to_s
if(break_flag==1)
  puts 'break'
end

```

```

puts 'pp = '+pp.to_s
filename_contents = 'contents.txt'
contents_array = []

contents_array.push title+"\n"
contents_array.push Time.new
contents_array.push "\n"
contents_array.push "\n"

contents_array.push 'length = '+$length.to_s+"\n"
contents_array.push 'atom_hei = '+$atom_hei.to_s
+ "\n"
contents_array.push 'sol_hei = '+$sol_hei.to_s+"\n"
contents_array.push 'height = '+$height.to_s+"\n"
contents_array.push 'solventwidth = '
+$solventwidth.to_s+"\n"
contents_array.push 'time = '+$time.to_s+"\n"
contents_array.push 'select = '+$select.to_s+"\n"
contents_array.push "\n"
contents_array.push 'seedlimit = '+$seedlimit.to_s
+ "\n"
contents_array.push 'feedlimit = '+$feedlimit.to_s
+ "\n"
contents_array.push 'first_seedfront =
  ['+first_seed.join(', ')+']'+ "\n"
contents_array.push 'first_feedfront =
  ['+first_feed.join(', ')+']'+ "\n"
contents_array.push "\n"
contents_array.push print_front()
contents_array.push "\n"
contents_array.push 'feedfront = ['+$feedfront.join(', ')
+']'+ "\n"

```

```

contents_array.push 'seed_growth = '
+($seedfront[$seedfront.length-1][1]
-first_seed[0][1]).to_s
+"\n"
contents_array.push 'after_solventwidth = '
+($feedfront[1]-$seedfront[$seedfront.length-1][1]).to_s
+"\n"
contents_array.push 'solute_average = '
+(solute_ave/$time).to_s+"\n"
contents_array.push 'solute最終値 = '
+($solute.length).to_s+"\n"
contents_array.push "\n"
contents_array.push '[ ledge : edge ] = [ '
+$ledge_count.to_s+' : '+$edge_count.to_s+' ]'++"\n"
contents_array.push 'first_count0 = '+ (first_count0).to_s
+"\n"
contents_array.push 'end_count0 = '+count0().to_s+"\n"
contents_array.push '実時間 = '+ (t1-t0).to_s
contents_array.push "\n"
contents_array.push 'CPU時間 = '
+(tms1.utime-tms0.utime).to_s
contents_array.push "\n"
if(break_flag==1)
  contents_array.push 'break'++"\n"
end

File.open filename_contents,'w' do |f|
  f.write ''
end

File.open filename_contents,'a' do |f|
  f.write contents_array
end

end

```


A.1.2 autocalc-mse2d.rb

autocalc-mse2d.rb

```
require 'mse2d'

title = 'new_Normal_Feed'
autocalc = 1 #<---calc times
path = Dir.pwd

puts 'START'
for i in 0..autocalc-1

  $length = 300
  $atom_hei = 150
  $sol_hei = 150
  $height = $atom_hei+$sol_hei
  $solventwidth = 100
  $feedlimit = $height/2
  $seedlimit = 0# $height/10
  $feedfront = []
  $seedfront = []
  $seed_end = []
  center = $length/2
  width = $solventwidth/2
  front_move = (center-width-1)/2
  $feedfront = [$height-1,center+width-front_move]
  $seedfront[0] = [$atom_hei-1,center-width-1-front_move]
  $seed_end = [$atom_hei-1,0]
  $time = 10000
  $select = 1000
```

autocalc-mse2d.rb

```
#dir_name = title+"_len"+$length.to_s
+ "_hei"+$height.to_s+"_sol"+$solventwidth.to_s
+ "_select"+$select.to_s+'_time'+$time.to_s
#dir_name = "sample"
dir_name = "present"
Dir.chdir(path)
Dir.mkdir(dir_name)
ARGV[0]=dir_name

puts
puts title+' : mse2d ---> '+ARGV[0]
mse2d(title)

end
puts 'END'
```

A.2 Mandelbrot 集合計算プログラム

Mandelbrot.rb

```
require 'complex'

def Mandelbrot(x,y)
  count=20
  z0=Complex.new(x,y)
  z=z0
  while (z.abs<=4.0)&&(count>0) do
    z=z*z-z0
    count-=1
  end
  return count
end

num = gets.chomp.to_i
y=-2.0
dd=1.0/num

for time in 0..3
  while (y<=2.0) do
    x=-1.0
    while (x<=2.5) do
      printf("%10.5f, %10.5f, %3d\n",x,y,Mandelbrot(x,y));
      x+=dd
    end
    printf("\n")
    y+=dd
  end
  y=-2.0
  puts 'end ' + "\n"
end
```

A.3 QuickTime ファイルへの書き出し

シミュレーション結果保存において，実行結果を QuickTime ファイルへ書き出すための手順を記す．

1. まず，連番イメージから QuickTime ファイルへ書き出すフリーウェア，QTexport をダウンロードする．

mac 版 (http://www.zaw.jp/download/qt_m.cgi)

windows 版 (http://www.zaw.jp/download/qt_w.cgi)

エンコード自体は QuickTimePro と全く同じなので，無償で Pro の代わりとして使える．

2. アニメーションを行うと自動的に `made save.eps.gpl` というファイルが作成されているので，gnuplot で実行することにより eps ファイルとして保存する．

3. eps ファイルではエンコードが不可能なため，次ページの `change-gif.rb` を実行し，gif への変換を行う．

4. 作成された gif ファイルを QTexport でエンコードし，QuickTime ファイルとして書き出す．

```

Dir.mkdir("jpeg")
Dir.mkdir("gif")
Dir.mkdir("eps_point")
Dir.mkdir("dat_point")
for i in 0..100
  if(i<10)
    number = '000'+i.to_s
  elsif(i>=10 && i<100)
    number = '00'+i.to_s
  elsif(i>=100 && i<1000)
    number = '0'+i.to_s
  else
    number = i.to_s
  end
  a = 'gs -dNOPAUSE -sDEVICE=jpeg -sOutputFile=res_point'+
    +number+'.jpg -q -dBATCH -g500x300 res_point'+number+'.eps'
  %x[#{a}]
  b = 'sips -s format gif res_point' + number
    + '.jpg --out res_point' + number + '.gif'
  %x[#{b}]
  c = 'mv -f -v res_point'+ number + '.jpg ... jpeg_point'
  %x[#{c}]
  d = 'mv -f -v res_point'+ number + '.gif ... gif_point'
  %x[#{d}]
  e = 'mv -f -v res_point'+ number + '.eps ... eps_point'
  %x[#{e}]
  f = 'mv -f -v res_point'+ number + '.dat ... dat_point'
  %x[#{f}]
  puts 'change res_point'+number+'.eps'
end

```