mao manual

佐藤晴郎

2003.2.17

1　mao とは

mao とはこの論文の筆者である佐藤晴郎が修士論文の題材として設計したプログラミング言語のことであり、またその設計に基づき製作した精度保証付き数値計算が可能な処理系の事である。

2　命名理由

本処理系を実際に動かしてみると「正しい答え」を出力していると思われている (思いたい) コンピュータにもこれだけの誤差 (間違い) があるのだと実感することができる。さて、日本のことわざには「どのように優れたものも時には誤ることもある。」の意味で「弘法も筆の誤り」という言葉がある。本処理系の命名は、そのことわざから、始め「koubou」にしようと考えていた。しかし本処理系の開発日数はまだまだ浅く、誕生したばかりで、改善・発展の余地があることから「弘法大師 (空海)」の幼名、「真魚」から「mao」と名付ける事とした。

3　mao の実行

この節では、Windows 環境に限定して話を進めていく。Windows 環境で mao を起動するには、Cygwin がインストール済みであることが必要である。Cygwin とは、Cygnus Solutions が製作したもので一般的な GNU の開発ツールを含む UNIX のさまざまなツールを Windows で動作できるようにするものである。参考までに 2003 年 1 月現在の公式ウェブページを記述しておく。

`http://www.cygwin.com/`

mao の実行は mao.exe をダブルクリックで起動する。もしくは、コマンドプロンプト上で mao.exe を指定すればよい。

```
% mao.exe
```

また、Cygwin のシェル (パイプ)[1] や cat を利用してファイルからの入力をすることも出来る。

```
% cat 入力ファイル | mao.exe
or
% mao.exe < 入力ファイル
```

mao を起動すると “>”のプロンプトが現れる。

mao の入力はそのプロンプトの後に書く。ほとんどの場合、入力の後に改行することでプログラム (演算) が実行され、つぎの行にその結果が出力される。

4 mao での入力作業

mao では入力作業を効率化するため、GNU Readline ライブラリをもちいており、同ツールの機能を利用することが出来る。最低限の操作を以下にあげておくことにする。この操作は多くの環境で実行できるであろう。しかし初期設定を変更した場合など、上手く動かない場合もある。そのことは十分に承知しておいてほしい。

- →: 一つ先にカーソルを進める
- ←: 一つカーソルを戻す
- ↑: 一つ前に入力した文を呼び出す
- ↓: 一つ後の入力文を呼び出す
- Backspace: カーソルの一字前を削除する
- Delete: カーソル位置の文字を削除する
- tab: ファイル名の補完

その他、GNU Readline ライブラリの機能については、簡単ではあるが付録**??**に記述しておくので、そちらも参照してほしい。

[1] UNIX で用いられている用語

5　値の入力

```
> 1234
1.23400000000000 × 10^{3}

> 0.25
[ 2.49999999999999 × 10^{-1} @ 2.50000000000000 × 10^{-1} ]

> .5
[ 4.99999999999999 × 10^{-1} @ 5.00000000000000 × 10^{-1} ]

> 0.1
[ 9.99999999999999 × 10^{-2} @ 1.00000000000001 × 10^{-1} ]
```

6　区間の入力

区間も値として扱うことができる。mao では区間の間を “,” で区切るところを “@” で表す。具体的な入力方法は次のようにする。

```
> [1234@2345.67]
[ 1.23400000000000 × 10^{3} @ 2.34567000000001 × 10^{3} ]
```

7　変数

```
> a=12345

> a
1.23450000000000 × 10^{4}

> b=23456

> a+b
3.58010000000000 × 10^{4}
```

変数への代入は “=” の左に変数名、右に値を入れる。変数の呼び出しは変数名を書けばよい。

8　基本的な演算子

和、差、積、商、括弧、単項マイナスを使用できる。

以下に例を示す。

```
> 1+2-3*4+6/3
-7.00000000000000 × 10^{0}

> ((1+2)-10+20)*3
```

```
3.90000000000000 × 10^{1}

> -123
-1.23000000000000 × 10^{2}
```

9　関係演算子、論理演算子

関係演算子の結果はその関係が真ならば $[1,1]$、関係が偽ならば $[0,0]$ を返している。

```
> 1<2
1.00000000000000 × 10^{0}

> 12.3>20
0

> 10*10 <= 100
1.00000000000000 × 10^{0}

> 10*10 >= 100
1.00000000000000 × 10^{0}

> 1==2
0

> [1@2]==[2@4]
0

> [1@2]!=[2@4]
1.00000000000000 × 10^{0}

> [1@4] << [0@5]
1.00000000000000 × 10^{0}

> [1@4] >> [0@5]
0

> [1@4] <<= [1@4]
1.00000000000000 × 10^{0}

> [1@4] >>= [0@5]
0
```

上から順に $<$, $>$, $\leq$, $\geq$ は不等式の意味

==は等号、!=は非等号

<<,>>,<<=,>>=は包含関係の意味であり、先、二つは完全に含むの意味である。後、二つは $\supset$, $\subset$ を示す。

その他、論理演算子として&&,||,!を使用できる。

10　組み込み関数

組み込み関数として以下のものを利用できる。

- **sin(x)** : $\sin(x)$
- **cos(x)** : $\cos(x)$
- **tan(x)** : $\tan(x)$
- **cot(x)** : $\cot(x)$
- **sinh(x)** : $\sinh(x)$
- **cosh(x)** : $\cosh(x)$
- **tanh(x)** : $\tanh(x)$
- **coth(x)** : $\coth(x)$
- **asin(x)** : $\arcsin(x)$
- **acos(x)** : $\arccos(x)$
- **atan(x)** : $\arctan(x)$
- **acot(x)** : $\mathrm{arccot}(\mathrm{x})$
- **exp(x)** : e^x
- **log(x)** : $\log_e(x)$
- **log10(x)** : $\log_{10}(x)$
- **abs(x)** : $|x|$
- **sqr(x)** : x^2
- **sqrt(x)** : $\sqrt{x}$

11　組み込み定数

組み込み定数として以下 5 つのものを用意する。そしてそれを利用するには定数名をかけばよい。

```
> DEG
[ 5.72957795130823 × 10^{1} @ 5.72957795130824 × 10^{1} ]

> E
[ 2.71828182845904 × 10^{0} @ 2.71828182845905 × 10^{0} ]

> GAMMA
[ 5.77215664901531 × 10^{-1} @ 5.77215664901533 × 10^{-1} ]

> PHI
[ 1.61803398874989 × 10^{0} @ 1.61803398874990 × 10^{0} ]
```

```
> PI
[ 3.14159265358979 × 10^{0} @ 3.14159265358980 × 10^{0} ]

>
> PHI+PI
[ 4.75962664233968 × 10^{0} @ 4.75962664233969 × 10^{0} ]
```

12 行列

行列の入力は”[”“]”で囲い、列の区切りには“,”、行の区切りには“;”を用いる。また、行列の演算 $(+,\ -,\ \times,\ '$(転置)$)$、変数への代入をサポートしている。以下にその入出力例を示す。

```
> [1,2;3,-4]+[2,3;4,5]
( 3.00000000000000 × 10^{0} 5.00000000000000 × 10^{0} )
( 7.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )


> A=[1,2;3,-4]+[2,3;4,5]

> A
( 3.00000000000000 × 10^{0} 5.00000000000000 × 10^{0} )
( 7.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )


> A'
( 3.00000000000000 × 10^{0} 7.00000000000000 × 10^{0} )
( 5.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )


> [PI,[1@2];3,-4]+[2,[1@2];4,[0@2]]
( [ 5.14159265358979 × 10^{0} @ 5.14159265358980 × 10^{0} ] [ 2.00000000000000
× 10^{0} @ 4.00000000000000 × 10^{0} ] )
( 7.00000000000000 × 10^{0} [ -4.00000000000000 × 10^{0} @ -2.00000000000000
× 10^{0} ] )


> [1,2;3,-4;2,2]+[2,3;4,5;1,1]
( 3.00000000000000 × 10^{0} 5.00000000000000 × 10^{0} )
( 7.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )
( 3.00000000000000 × 10^{0} 3.00000000000000 × 10^{0} )


> [1,2;2,3]*[1,2;3,3]
( 7.00000000000000 × 10^{0} 8.00000000000000 × 10^{0} )
( 1.10000000000000 × 10^{1} 1.30000000000000 × 10^{1} )
```

```
> [1,2,2;3,1,2;1,1,1]*[1,2;3,1;1,1]
( 9.00000000000000 × 10^{0} 6.00000000000000 × 10^{0} )
( 8.00000000000000 × 10^{0} 9.00000000000000 × 10^{0} )
( 5.00000000000000 × 10^{0} 4.00000000000000 × 10^{0} )


> [1,1,1]*[1,2,2;3,1,2;1,1,1]
( 5.00000000000000 × 10^{0} 4.00000000000000 × 10^{0} 5.00000000000000
× 10^{0} )


> [1,2,2;3,1,2;1,1,1]*[1;1;1]
( 5.00000000000000 × 10^{0} )
( 6.00000000000000 × 10^{0} )
( 3.00000000000000 × 10^{0} )
```

13 行列の要素単位での操作

行列の要素単位での操作
詳細については第**??**節 mao 言語仕様を参考にしてもらいたい。

```
> a=[1,2,3;4,5,6;7,8,9]

> a[1,1]=3

> a
( 3.0000 × 10^{0} 2.0000 × 10^{0} 3.0000 × 10^{0} )
( 4.0000 × 10^{0} 5.0000 × 10^{0} 6.0000 × 10^{0} )
( 7.0000 × 10^{0} 8.0000 × 10^{0} 9.0000 × 10^{0} )


> b=[1,1;1,1]

> a[1,1]=b

> a
( 1.0000 × 10^{0} 1.0000 × 10^{0} 3.0000 × 10^{0} )
( 1.0000 × 10^{0} 1.0000 × 10^{0} 6.0000 × 10^{0} )
( 7.0000 × 10^{0} 8.0000 × 10^{0} 9.0000 × 10^{0} )


> a[1,1]+=b

> a
( 2.0000 × 10^{0} 2.0000 × 10^{0} 3.0000 × 10^{0} )
( 2.0000 × 10^{0} 2.0000 × 10^{0} 6.0000 × 10^{0} )
( 7.0000 × 10^{0} 8.0000 × 10^{0} 9.0000 × 10^{0} )
```

```
> a[2,1]-=b

> a
( 2.0000 × 10^{0} 2.0000 × 10^{0} 3.0000 × 10^{0} )
( 1.0000 × 10^{0} 1.0000 × 10^{0} 6.0000 × 10^{0} )
( 6.0000 × 10^{0} 7.0000 × 10^{0} 9.0000 × 10^{0} )


>
> a[2:3,2:3]
( 1.0000 × 10^{0} 6.0000 × 10^{0} )
( 7.0000 × 10^{0} 9.0000 × 10^{0} )


> a[2:3,:]
( 1.0000 × 10^{0} 1.0000 × 10^{0} 6.0000 × 10^{0} )
( 6.0000 × 10^{0} 7.0000 × 10^{0} 9.0000 × 10^{0} )
```

14 特別な行列を作る

行列を作る命令として eye、zeros、ones を用意した。

```
> eye(2,3)
( 1.00000000000000 × 10^{0} 0 )
( 0 1.00000000000000 × 10^{0} )
( 0 0 )


> zeros(2,2)
( 0 0 )
( 0 0 )


> ones(2,2)
( 1.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )
( 1.00000000000000 × 10^{0} 1.00000000000000 × 10^{0} )


> eye(4)
( 1.00000000000000 × 10^{0} 0 0 0 )
( 0 1.00000000000000 × 10^{0} 0 0 )
( 0 0 1.00000000000000 × 10^{0} 0 )
( 0 0 0 1.00000000000000 × 10^{0} )
```

15 print

出力は print 文で作られる。print 文の引数は、C と同じように二重引用符でくくった文字列、あるいは式をコンマで区切った並びを使う。また改行などを表す記号としてバックスラッシュと英小文字を組み合わせたエスケープシーケンス (拡張表記) を用いることが出来る。それは以下に示すとおりである。

\ a	アラート	聴覚的な警報を発する。
\ b	後退	表示位置を直前に移動する。
\ t	水平タブ	次の水平タブ位置へ移動する。
\ n	改行	改行して次の行の先頭へ移動する。
\ r	復帰	現在の行の先頭位置へ移動する。

例えば print 文の入力は次のような形で書く。

```
> print("abc","\t", 123, "\n","def")
abc 1.23000000000000 × 10^{2}
def
```

16 制御フロー

制御フローとして if、if-else、while を用いることが出来る。しかし C 言語と異なり break 文、continue 文、for 文はない。mao ではセミコロンが特別な意味をもたない。文と文は改行で区切られる。入出力例は以下の通りである。

```
> x=1

> if(x>0) print("Yes\n") else print("no\n")
Yes

>
> if(x<0) print("Yes\n") else print("no\n")
no

>
> if(x<0){

> print("Yes\n")

> }else{

> print("No\n")
```

```
> }
No

>
> while(x<10){

> print(x, "\n")

> x=x+1

> }
1.00000000000000 × 10^{0}
2.00000000000000 × 10^{0}
3.00000000000000 × 10^{0}
4.00000000000000 × 10^{0}
5.00000000000000 × 10^{0}
6.00000000000000 × 10^{0}
7.00000000000000 × 10^{0}
8.00000000000000 × 10^{0}
9.00000000000000 × 10^{0}
```

後半二つの入力には中括弧が必須である。もしこの中括弧がなければ、構文エラーとなる。

17 関数と手続き

mao ではユーザーが関数や手続きを作ることが出来る。関数と手続きの違いは関数が値を返し、手続きは値を返さない。上がみたされない場合はエラーとなる。関数と手続きには起動するときにコンマで区切った引数を書き加えても良い。引数は関数または手続き内で参照することができ、$2 と書けば 2 番目の引数という意となる。関数と手続きは再帰的にもちいてもよい。

以下に入出力例を示す。

```
> proc aa(){

> print(12,"\t",39,"\n")

> }

>
> aa()
1.20000000000000 × 10^{1} 3.90000000000000 × 10^{1}

>
> proc sum(){
```

```
> i=0

> j=0

> while(i<$1){

> j=j+i

> i=i+1

> }

> print(j,"\n")

> }

>
> sum(10)
4.50000000000000 × 10^{1}

> sum(100)
4.95000000000000 × 10^{3}

>
> func bb(){

> return($1*5)

> }

>
> bb(10)
5.00000000000000 × 10^{1}

>
> func cc(){

> return($1*bb(10))

> }

>
> cc(2)
1.00000000000000 × 10^{2}
```

18　ファイルの読み込み

mao ではあらかじめ作っておいたファイルからの読み込み (入力) をすることが出来る。
入力方法は

```
> load（" ファイル名 "）
or
> load " ファイル名 "
```

とする。

19　変数、関数、手続きへの操作

自分の設定した変数、関数、手続きの確認をすることが出来る。入力方法は下のようにする。

```
> whos
```

設定した変数は上書き可能だが、関数、手続きは上書きすることができない。このことから、変数、関数、手続きを削除することを可能にした。実行方法は

```
> free（変数名、ユーザー定義関数名、ユーザー定義手続き名）
```

また全ての変数、関数、手続きを削除するには

```
> free
```

とすればよい。上を行うと、mao では記憶している変数、関数、手続きのメモリを削除し、さらにそれ以外にも不要となったメモリを解放する。よって、これはメモリを節約するためにも効果的な方法である。

20　LU 分解

mao では LU 分解に対する命令を持っている。
正方行列 A の LU 分解とは

$$A = LU$$

ただし L は下三角行列、U は上三角行列 と分解することである。

次に部分ピボット交換ありの LU 分解とは正方行列 A を以下の様にすることである。

$$PA = LU$$

P は置換行列、L は下三角行列、U は上三角行列である。

mao での入力形式は

```
> [ VAR1 VAR2 ] = lu( expr )

> [ VAR1 VAR2 VAR3] = lu( expr )
```

VAR1、VAR2、VAR3 は変数名である。

expr には LU 分解をしたい正方行列、もしくは LU 分解をしたい正方行列を代入した変数を入力する。

一つ目の命令は VAR1 に L、VAR2 に U を代入する。二つ目の命令は VAR1 に L、VAR2 に U、VAR3 に P を代入する。

入出力例は以下のとおりである

```
> a=[1,2,3;4,4,4;5,6,8]

> [ lower upper ] = lu(a)

> lower
( 1.00 × 10^{0} 0 0 )
( 4.00 × 10^{0} 1.00 × 10^{0} 0 )
( 5.00 × 10^{0} 1.00 × 10^{0} 1.00 × 10^{0} )


> upper
( 1.00 × 10^{0} 2.00 × 10^{0} 3.00 × 10^{0} )
( 0 -4.00 × 10^{0} -8.00 × 10^{0} )
( 0 0 1.00 × 10^{0} )


> a-lower*upper
( 0 0 0 )
( 0 0 0 )
( 0 0 0 )


> [ lower upper pivot ] = lu(a)

> lower
( 1.00 × 10^{0} 0 0 )
( [ 1.99 × 10^{-1} @ 2.01 × 10^{-1} ] 1.00 × 10^{0} 0 )
( [ 7.99 × 10^{-1} @ 8.01 × 10^{-1} ] [ -1.01 × 10^{0} @ -9.99 ×
10^{-1} ] 1.00 × 10^{0} )
```

```
> upper
( 5.00 × 10^{0} 6.00 × 10^{0} 8.00 × 10^{0} )
( 0 [ 7.99 × 10^{-1} @ 8.01 × 10^{-1} ] [ 1.39 × 10^{0} @ 1.41 ×
10^{0} ] )
( 0 0 [ -1.01 × 10^{0} @ -9.99 × 10^{-1} ] )


> pivot
( 0 0 1.00 × 10^{0} )
( 1.00 × 10^{0} 0 0 )
( 0 1.00 × 10^{0} 0 )


>
> pivot * a - lower * upper
( 0 0 0 )
( [ -Epsilon @ 1.12 × 10^{-16} ] [ -4.45 × 10^{-16} @ 4.45 × 10^{-16} ]
[ -4.45 × 10^{-16} @ 4.45 × 10^{-16} ] )
( [ -8.89 × 10^{-16} @ 4.45 × 10^{-16} ] [ -2.67 × 10^{-15} @ 2.67
× 10^{-15} ] [ -6.22 × 10^{-15} @ 5.78 × 10^{-15} ] )
```

21 逆行列

逆行列を求めるには

```
> a=[1,1,1;1,2,2;1,2,3]

> inv(a)
( 2.00 × 10^{0} -1.00 × 10^{0} 0 )
( -1.00 × 10^{0} 2.00 × 10^{0} -1.00 × 10^{0} )
( 0 -1.00 × 10^{0} 1.00 × 10^{0} )
```

22 連立一次方程式を解く

連立一次方程式

$$Ax = b \quad (A, b \in expr)$$

を解くには

```
> A \ b
```

とする。この命令は mao の内部で A についての部分ピボット変換ありの LU 分解をしてから連立一次方程式を解いている。

23　最大値、最小値

区間の最大値、最小値を求められる。mao での入力方法は

```
> max(expr)    (or min(expr) )
```

行列の場合もこのコマンドは有効で各要素に対して上を実行する。

24　絶対値

絶対値を求めることができる。mao での入力方法は

```
> | expr |    (or abs(expr) )
```

行列の場合は各要素に対して絶対値をとる。

25　最大値ノルム

最大値ノルムを求めることができる。mao での入力方法は次の様にする。

```
> norm(expr)
```

26　設定の変更

mao では幾つかの設定を変更することが出来る。それを以下にあげる。

26.1　桁数

まず、出力の桁数を指定することが出来る。 (現段階では 1 から 15 まで) 指定の仕方は

```
> mode( precision : 数 )
or
> precision = 数
```

precision は ketasuu との入力でもよい。

26.2　区間の入力に対するチェック

区間の入力の際のチェックに対する動作の指定は

```
> mode( iicheck : A )
or
> iicheck = A
```

(ただし A は strict、none、warning、stop、default、kihon の何れか)

- strict,default,kihon: 区間のチェックを通常通り行う。
- none: 区間のチェックをせず、計算を行う。
- warning: 通常の区間のチェックを行い、ユーザーに確認を求める動作が生じた際は画面にその旨を表示し、以後の計算を続行する。
- stop: 通常の区間のチェックを行い、ユーザーに確認を求める動作が生じた際はそこで計算を打ち切る。

iicheck とは interval input check の略である。また iicheck は kukaninput との入力でもよい。

26.3 丸めの指定

丸めに対する動作の指定は

```
> mode( rounding : A )
or
> rounding = A
```

(ただし A は default、kihon、up、down、near の何れか)

- default, kihon: 通常通り精度保証が出来る形で丸めの方法を実行する。
- up: 丸めの方法を常に上向きの丸めとして計算する。
- down: 丸めの方法を常に下向きの丸めとして計算する。
- near: 丸めの方法を常に最近点への丸めとして計算する。

26.4 設定変更についての以後の課題

本論文提出時には間に合わせることが出来なかったが近似モード、誤差の表示モード (半径での表示を可能にする) を変更できれば尚、良かったと思われる。

27 シェルモード

以下の入力をすることでシェルでの命令 (コマンド) を起動することが出来る。

```
> _ シェルで入力する命令（例えば ls 等）
```

使えるコマンドはシステムによって異なるが C 言語における system 関数で呼び出せるものとする。

28 plot モード

以下の入力をすることで gnuplot 等で応用が出来るファイルを任意のファイルに保存することが出来る。

```
> plot ( "ファイル名", expr )
```

expr がスカラー区間の場合。すなわち $expr = [a@b]$ の場合。出力ファイルの中身は次のようになっている。

$$a\ b$$

a と b の間はスペースで区切られる。

expr が行列の場合。出力ファイルは

- 出力の 1 行 1 列目に、行列の 1 行 1 列目の区間の最小値
- 出力の 1 行 2 列目に、行列の 1 行 1 列目の区間の最大値
- 出力の 1 行 3 列目に、行列の 1 行 2 列目の区間の最小値
- 出力の 1 行 4 列目に、行列の 1 行 2 列目の区間の最大値
- ···
- 出力の 2 行 1 列目に、行列の 2 行 1 列目の区間の最小値
- 出力の 2 行 2 列目に、行列の 2 行 1 列目の区間の最大値
- 出力の 2 行 3 列目に、行列の 2 行 2 列目の区間の最小値
- 出力の 2 行 4 列目に、行列の 2 行 2 列目の区間の最大値
- ···

となる。具体的な応用例として

$$[1@2] \times \sin(x)$$

のグラフを書いてみる。まず mao での入力を次のようにする。

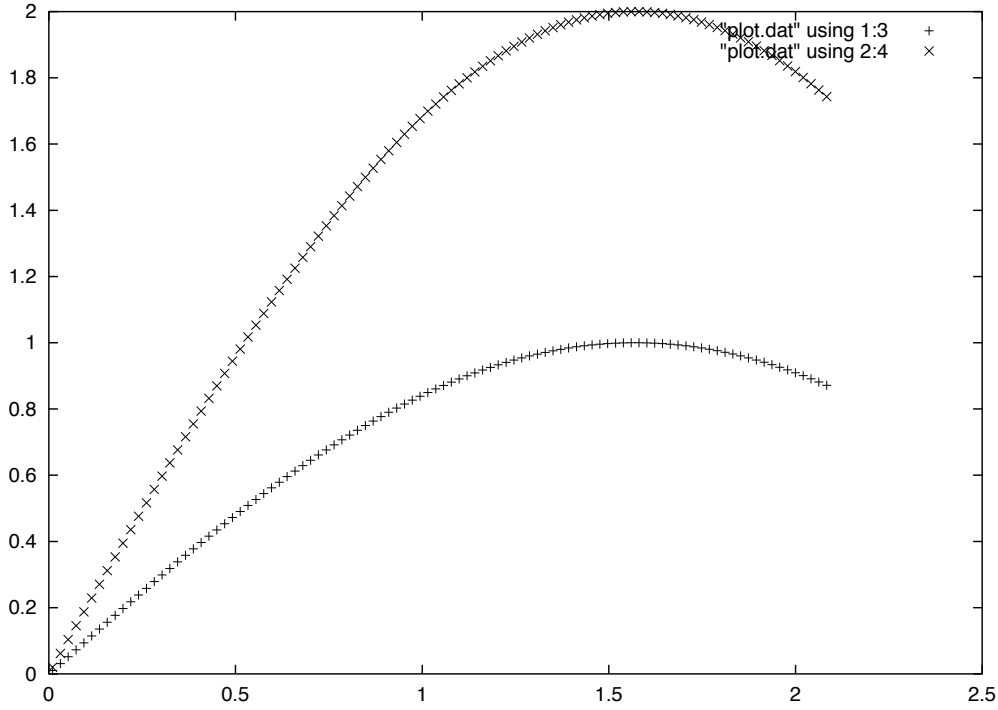
```
> a=zeros(100,2)

>
> i=1

> di=PI/150

> d=0.01
```

```
>
> while(i<=100){

> a[i,1] += d

> a[i,2] += [1@2] * sin(a[i,1])

> d+=di

> i=i+1

> }

> plot("plot.dat",a)
```

次に gnuplot で下のような命令を入力する。

```
gnuplot> plot "plot.dat" using 1:3 ,\
> "plot.dat" using 2:4
gnuplot>
```

すると、次のような結果が得られる。


"plot.dat" using 1:3
"plot.dat" using 2:4
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5