微分方程式は方向場と数値解で解く
－J による微分方程式グラフィックス－

西川　利男

１．「微分方程式を解く」とは

微分方程式は物理、化学はもちろん、経済学においても強力な武器となっている。
ところで、一般の数学の教科書で、微分方程式はどう教えられているだろうか？

常微分方程式では、変数分離、線形などと、微分方程式の式の形を見て、逆演算である積分により解く。これは頭の体操としては良いかも知れないが、まるでパズルである。
一般的な解法とは言えず、発展性もない。

もっと、微分方程式の式を良く観察したら良い。

$\frac{dy}{dx}=\frac{y}{x}$　は点(x, y)の各位置の勾配に等しい。つまり原点を通る直線である。

一方、

$\frac{dy}{dx}=-\frac{x}{y}$　はこれに直交する勾配であり、円に他ならない。

それでは、微分方程式はどうやって解くか。微分とは点の位置(x, y)における勾配である。これを図の各点の位置でそのまま表示したらよい。こうやって出来た図が方向場グラフィックスである。どういう式で表されるかは問わない。言うなれば積分はいらない。

次にある１点(x_0, y_0)を通るようなものはどうなるだろうか。これには隣りの数点の値を調べ、Runge Kutta の方法を代表とする数値計算を行い値を得て、これを曲線としてグラフィック画面に表す。これはふつうは特殊解あるいは初期値問題といわれる。私に言わせれば、こんなものものしい言い方ではなく、個別解と言えばよいものである。

なお、このテーマについては、数年前に数回にわたってすでに発表した [1], [2] 。今回の発表は最新のコンピュータ環境に合わせて、実行できるよう修正、改良したものである。

[1]　西川利男「微分方程式を APL/J 思考から理解する－微分方程式は４次元を３次元に映す鏡である－」J 研究会資料 2006/9/30

[2]　西川利男「J による微分方程式のグラフィック・アプローチーその１」
１階常微分方程式－数値解と方向場表示の活用」J 研究会資料 2006/10/28

２．J を動かす環境－Windows 7 上での J9 の不便さ

先に私がこのグラフィックシステムを開発した時点では、コンピュータ環境は Windows XP 上で J4 でプログラミングをおこなった。

ところが、多くの人から現在のコンピュータ環境の上で動くようにしてほしいという要望がある。今回、志村氏から J9 の提供を受け、私の手元の Windows 7 上で走らせ、プログラムの修正、書き換えを行った。その経過、結果を報告する。

なお、Windows 7 上で J4 を走らせて、先のプログラムを実行するとそのまま実行できた。

以下に Windows 7 上での J9 の問題点を上げる。

・Windows 操作の基本であったフォームからのボタン実行が J9 ではサポートされない。

Windows XP 上の J4 では、プログラムを実行して、そのフォームの入力編集ボタンにより方程式や値を入力し、実行ボタンを押して演算処理をおこない図示する。

ところが、J9 ではフォームエディタが廃止されてしまった。

したがって、プログラムの実行時に値を入れるなどという操作はできない。実行前にあらかじめ行わざるをえない。これはフォームエディタの廃止に伴う不便さであるが、これでは Windows プログラムの意味がなくなってしまう。

このため、J4 のときとは異なるユーザ・インターフェースで対処することにした。
プログラムの左引数として微分方程式、右引数として初期値をとり実行する。

例えば、次の微分方程式

$$\frac{dy}{dx} = (\cos x) - y$$

で方向場画面の上に、初期値を(0, 0), (0,1), (0,2) とする個別解を描きたいときには、

‘ (cos x) – y ’ run (0, 0);(0,2);(0, 3)

のように入力する。

・正規表現の機能は従来通り、動いている。

左引数の cos 部分は正規表現を用いて、2&o. と変換される。.

・J での唯一の救いは文字列からの動詞演算の機能は健在であった。

```
  B =: '1 + 3'
  $B    NB. 5 文字
5
  ". B    NB. 文字列を解釈して動詞として実行する
4
```

このような機能は他の言語にはなく、J の優れた特徴であり、今回のコーディングでも大いに活用している。

結局、J4 から J9 への移行では、プログラムのかなりの修正、見直しをすることになった。

３．Ｊプログラミングの概要

プログラムはかなり長い。概要は次のようである。詳細の説明は最後に示した。

- **J9 でのフォームの作成　…　フォームエディタによらない方法**
- **個別解－ルンゲクッタの方法**
- **方法場―矢印図形の表示**
- **方程式の通常表示からＪ表示への変換　…　正規表現を利用**

４．微分方程式－方向場と数値解グラフィックスシステムの実行の実際

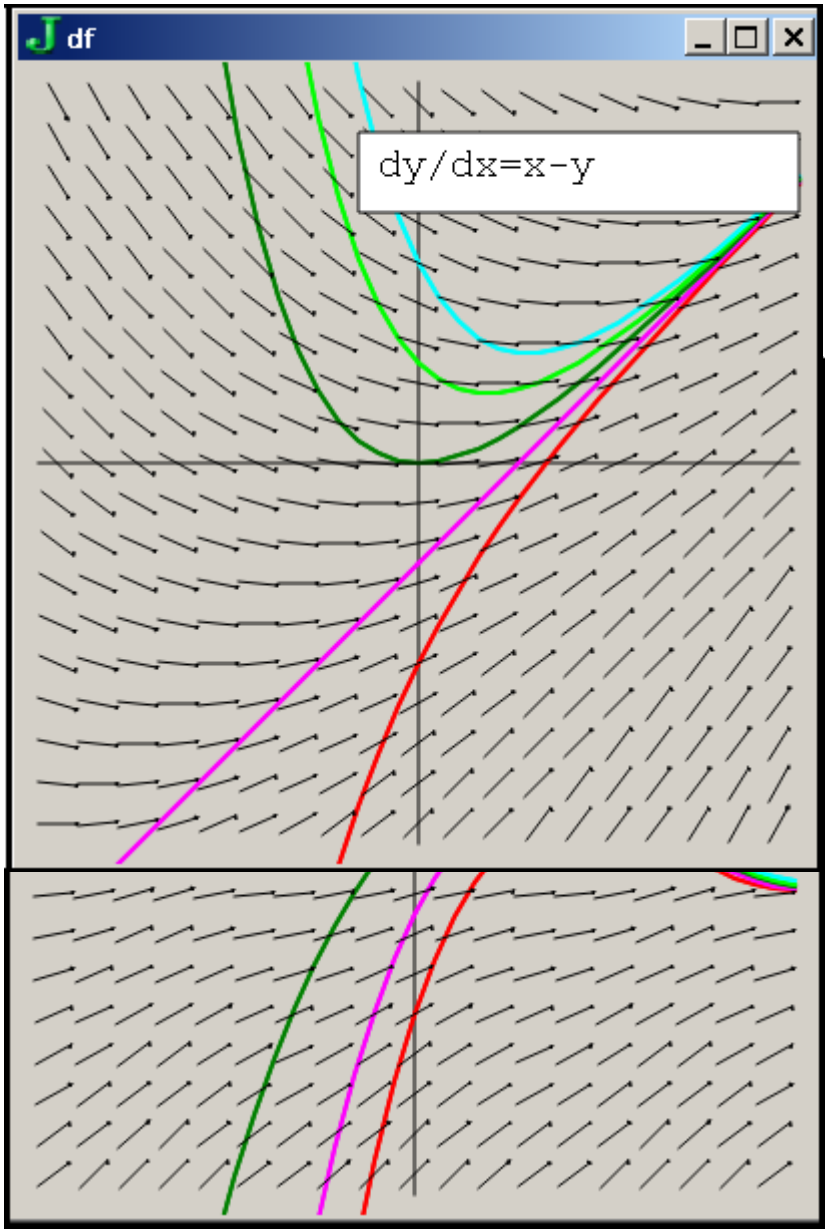
たとえば、このように入力すると、次のグラフィックスが表示される。

‘x - y’　run (0, _2) ; (0, _1) ; (0, 0) ; (0, 1) ; (0, 2)

つまり、微分方程式　$\frac{dy}{dx} = x - y$　の方向場グラフィックスとともに

初期値　(0, _2), (0, _1), (0, 0), (0, 1), (0, 2) を通るそれぞれの

個別解が色づけされて示される。



次に違う微分方程式でやってみる。

‘(cos x) - y’

run (0, _2) ; (0, _1) ; (0, 0) ; (0, 1) ; (0, 2)

結果はつぎのようになる。

５．あらためて微分方程式とは

従来のように積分ときにより式が得られれば、終わりとするのではなく、時間とともに移り行く現象の変化を観察する道

具として活用してもらいたい。

・小孔からの水の流出
・円筒形の液面の形
・大気圧と高さ
・細菌の増殖数
・放射性物質の壊変

など、いろいろな問題が微分方程式により、解明されている。[3]

[3] 西本勝之「初学者のための微分方程式」p. 116-158, 森北出版(1961).

６．微分方程式－方向場と数値解グラフィックスのJプログラミングの詳細

Jの実際プログラムをコーディングに沿って、解説した。

６．１　準備

```
load 'graph'
load 'numeric trig'
```

・J9 ではgl2グラフィックス命令を実行するに、従来の次の２行では出来ない。

```
NB. require 'gl2'
NB. require 'isigraph'
```

６．２　gl2グラフィックスのインストール

・coinsert により、クラスファイル jgl2 のインスタンスを導入して、はじめて gl2 グラフィックス命令が可能になる。

```
coinsert 'jgl2' NB. required for J9
```

６．３　gl2グラフィックスのフォームの作成と実行

・フォームエディタによらず、次の手順で手動操作で作らなくてはならない。
　フォームの名前は　DF、 df として作成した。

NB. make graphic form フォームの作成

```
DF=: 0 : 0
pc df closeok;
minwh 400 400;cc g isigraph flush;
pas 0 0;
)
Eq =: 'x - y'            NB. 微分方程式
```

NB. excute form フォームの実行プログラム

```
run=: df_run
df_run=: 3 : 0
Eq df_run y         NB. 左引数なしの場合はデフォルト関数 Eq をとる
:
wd DF
Eq =: x                      NB. 左引数、微分方程式の式
INIT =: y          NB. 右引数、初期値(ボックス)、デフォルトでは複数、
if. 0 = (#@$) > INIT    NB. 右引数が複数(ボックス)かどうかのチェック
 do.
   INIT =: INIT
 end.
```

```
if. 1 = (#@$) > INIT       NB. 右引数が１つだけの場合は、それをボックスにする。
 do.
       INIT =: < INIT
 end.
wr INIT
test0 "  NB. 個別解プログラムを実行
test2 "  NB. 方向場プログラムを実行
wd 'pshow'
)
```

６．４　グラフィックス表示

プログラムの起動とともに、df_g_paint が実行され、グラフィックスが表示される。

６．４．１　座標軸を引く

```
NB.  draw graphics ======
df_g_paint=: 3 : 0
gllines 10, 200, 390, 200 NB. x-axis  X 軸を引く
gllines 200, 10, 200, 390 NB. y-axis   Y 軸を引く
```

６．４．２　個別解プログラムの実行で得られた値を画面に表示

```
NB.  particular solutions / ====
j =. 0
while. j < #DA
 do.
     glrgb > j{ COL
     glpen 2
     gllines >j{ DA
     gllines >j{ DB
     glrgb > j{ COL
     glpen 2
     gllines >j{ DA
     gllines >j{ DB
     j =. j + 1
end.
```

６．４．３　方向場プログラムの実行で得られた値を画面に表示

```
NB. differential field / ======
glrgb 0 0 0
glpen 1
 i =. 0
 while. i < #TT
  do.
   gllines > i{TT
   i =. i + 1
  end.
```

６．４．４　微分方程式名を画面に表示

```
NB. label of differential equation / ======
glrgb 255 255 255       / 白抜きの長方形
glbrush ''
glrect 170 35 220 40
glrgb 0 0 0             / 黒字で微分方程式名を書く
gltextcolor ''
glfont '"courier new" 12'
gltextxy 180 40
gltext 'dy/dx=', Eq       / Eq は方程式名
)
```

例えば、

‘x - y’ run (0, _2);(0,_1);(0,0); (0,1);(0,2)

によって、上のすべてが実行されて、グラフィックス画像が表示される。

６．５　個別解の計算とルンゲ・クッタ法による数値解サブルーティン

６．５．１　初期値の各値に対して、個別解を画像値を計算する

```
NB. particular X, Y - solution  ========================
adjtest1 =: 3 : 0
, roundint (200, 200) +"(1) (50, _50) *"(1) y
)

test0 =: 3 : 0
DA =: "
DB =: "
j =. 0
while. j < (#INIT)
 do.
   DA1 =. 0.2 (df Eq) rk^:(i. 20) (> j{INIT)
   DA0 =. adjtest1 DA1
   DA =: DA, < DA0
   DB1 =. _0.2 (df Eq) rk^:(i.20) (> j{INIT)
   DB0 =. adjtest1 DB1
   DB =: DB, < DB0
   j =. j + 1
 end.
)
```

６．５．２　ルンゲ・クッタ法による数値解サブルーティン

```
NB. Solve Differential Equation ==================================
NB. Runge-Kutta argumented Dif_Equation /2006/9/5
NB.   inc (dif_eq)  rk^:(cycles) initial x, y
NB. Using d-verb, Enable Ordinary Notation as d 'y-x'
NB.   0.1 (df 'y-x') rk^:(i.11) 0 0
rk =: 1 : 0
:
h =. x
X =: 0{ y
Y =: 1{ y
F =: u
```

```
k1 =: ″. F, (″:X),',',(″:Y)
k2 =. ″. F, (″:X + h%2),',',(″:Y + h*k1%2)
k3 =. ″. F, (″:X + h%2),',',(″:Y + h*k2%2)
k4 =. ″. F, (″:X + h),',',(″:Y + h*k3)
(X+h), (Y + h*(k1 + (2*k2) + (2*k3) + k4)%6)
)
```

６．６　方向場の計算と矢印図形の計算

６．６．１　方向場の計算

```
NB. direction field calculation  ====================================
NB. point range X = _12 to 12, y = _12 to 12
Dtest2 =:  > , (<"(1)) (i: 9) ,."(0, 1) (i: 9)

NB. calculate dy/dx
Stest2 =: 0.1 * ({. - {:)"(1) Dtest2 NB. dy/dx = x - y.5

exfunc =: 3 : 0
AA =. y
'M N' =. $ AA
RR =. ''
i =. 0
while. i < ({. $ AA)
 do.
   AAA =.''. (": eq), ":(i{ AA)
NB.    wr (i{AA), AAA
   RR =. RR ,(i{AA), AAA
   i =. i + 1
 end.
(M, (>: N))$ RR
)

Stest2A =: 0.1 * 2 {"(1) exfunc Dtest2

Dtest =: Dtest2, "(1, 0) Stest2A

NB. adjust to graphic form
adjtest2 =: 3 : 0
, roundint (200, 200) +"(1) (2, _2)* "(1) y
)

NB. make arrow figures from x, y, dy/dx
NB. test2 Dtest => TT
test2 =: 3 : 0
```

```
TT =: ''
i =. 0
while. i < #Dtest
  do.
    TT =: TT, < adjtest2 10 * |: > 0.5 arrow i{Dtest
    i =. i + 1
  end.
)
```

６．６．２　矢印図形の生成

```
NB. Arrow Symbol
NB.  length arrow center slope
NB.   eg. 1 arrow _2 1 0.5
arrow =: 3 : 0
:
require 'plot'
d =. x
'x0 y0 a' =: y
NB. arrow figure
'p0x p0y' =: (-d), 0
'p1x p1y' =. d, 0
'p2x p2y' =. (0.9*d), (0.1*d)
'p3x p3y' =. (0.9*d), (_0.1*d)
'p4x p4y' =. d, 0
NB. plot (_4, 4, 4, _4, _4, p0x, p1x, p2x);(_4, _4, 4, 4, _4, p0y, p1y, p2y)
px =. p0x, p1x, p2x, p3x, p4x
py =. p0y, p1y, p2y, p3y, p4y
NB. rotate arrow
Cos =. 1 % %: 1 + *: a
Sin =. a % %: 1 + *: a
'xx yy' =: (2 2$Cos, (-Sin), Sin, Cos) (+/ . *) (px ,: py)
xx =. x0 + xx
yy =. y0 + yy
NB. plot (_4, 4, 4, _4, _4, xx);(_4, _4, 4, 4, _4, yy)
xx;yy
)
```

６．７　微分方程式の通常表示からJ表示への変換

例えば、cos x は　2　o. x のように、J で実行できるように変換する

```
df =: 3 : 0
require 'regex'
y =. y
if. '-' = {. y do. y =. '0 ', y end.
y =. '\([[:space:]]*-' ('(0'&,@(}.)) rxapply y  NB. '(-' => '(0-'
y =. '[[:digit:]]+\.*[[:digit:]]*' (,&'"_') rxapply y
y =. ('\/';'%') rxrplc y
y =. ('sqrt';'%:@') rxrplc y
y =. ('sin';'1: o. ') rxrplc y
y =. ('cos';'2: o. ') rxrplc y
y =. ('tan';'3: o. ') rxrplc y
y =. ('cot';'4: o. ') rxrplc y
y =. ('exp';'^@') rxrplc y
y =. ('log';'^.@') rxrplc y
y =. ('x';'{.') rxrplc y
y =. ('y';'{:') rxrplc y
'(', y, ')'
)


NB. Amend in String Program by T. Nishikawa
str2box=: 3 : 0
:
Z1=. x E. y
Z2=. (|. x) E. (|. y)
Z3=. 1, }: |. Z2
Z4=. Z1 +. Z3                          NB. revised to (+.) 2006/6/28
Z5=. Z4 <;.1 y
AM=: ((<,x) = Z5) # i. #Z5 NB. AM is global value
Z5
)

of=: str2box  NB. alias for string amend
```

```
NB. Usage f.g.
NB.    'pq' amdstr 'xyz' of 'abcxyzdefgxyzhi'
NB. abcpqdefgpqhi
NB. revised 2006/6/28, 7/3
NB.   'pq' amdstr 'xyz' of 'xyzdefgxyzhi'
NB. pqdefgpqhi
NB.   'pq' amdstr 'x' of 'xyzdefgxyzhi'
NB. pqyzdefgpqyzhi

amdstr=: 3 : 0
:
; (<,x) AM } y              NB. revised 2006/7/3
)
```

dfield9.ijs - 最終プログラム

```
NB. dfield9.ijs 2020/5/27, 6/22
NB.   J9 version on Windows7
NB.   revised several points  --   by T. Nishikawa
NB. dfield J4,J5 version 2006/11/2 by T. Nishikawa
NB.        revised from suggestion by Y. Nakano
NB. ODE Direction Field and Numerical Calculation

load 'graph'
load 'numeric trig'

NB. require 'gl2'
NB. require 'isigraph'
coinsert 'jgl2' NB. required for J9

NB. make graphic form
DF=: 0 : 0
pc df closeok;
minwh 400 400;cc g isigraph flush;
pas 0 0;
)

Eq =: 'x-y'
run=: df_run
df_run=: 3 : 0
Eq df_run y
:
wd DF
Eq =: x
INIT =: y
if. 0 = (#@$) > INIT
  do.
     INIT =: INIT
  end.
if. 1 = (#@$) > INIT
  do.
```

```
    INIT =: < INIT
  end.
eq =: df Eq
test0 "  NB. NB. particular solutions
test2 "  NB. differential field
wd 'pshow'
)

NB. draw graphics ===================================
df_g_paint=: 3 : 0
gllines 10, 200, 390, 200 NB. x-axis
gllines 200, 10, 200, 390 NB. y-axis
NB. particular solutions ====================
j =. 0
while. j < #DA
  do.
glrgb > j{ COL
glpen 2
gllines >j{ DA
gllines >j{ DB
glrgb > j{ COL
glpen 2
gllines >j{ DA
gllines >j{ DB
j =. j + 1
end.
NB. differential field ========================
glrgb 0 0 0
glpen 1
   i =. 0
   while. i < #TT
    do.
      gllines > i{TT
      i =. i + 1
    end.
```

```
NB. label of differential equation ============
glrgb 255 255 255
glbrush "
glrect 170 35 220 40
glrgb 0 0 0
gltextcolor "
glfont '"courier new" 12'
gltextxy 180 40
gltext 'dy/dx=', Eq
)

NB. Imported from dfield.js J3 version ================================
NB. Ordinary Differential Equation(ODE)
NB. Direction Field Graphic Display 2006/9/27
NB. and DE Numerical Solution       2006/9/28
NB. revised for any x, y ranges     2006/10/6

NB. Conversion 'y / x' => '(1&{) % (0&{)'
df =: 3 : 0
require 'regex'
y =. y
if. '-' = {. y do. y =. '0 ', y end.
y =. '\([[:space:]]*-' ('(0'&,@(}.)) rxapply y  NB. '(-' => '(0-'
y =. '[[:digit:]]+\.*[[:digit:]]*' (,&'"_') rxapply y
y =. ('\/';'%') rxrplc y
y =. ('sqrt';'%:@') rxrplc y
y =. ('sin';'1: o. ') rxrplc y
y =. ('cos';'2: o. ') rxrplc y
y =. ('tan';'3: o. ') rxrplc y
y =. ('cot';'4: o. ') rxrplc y
y =. ('exp';'^@') rxrplc y
y =. ('log';'^.@') rxrplc y
y =. ('x';'{.') rxrplc y
NB. y =. ('x';'(0&{)') rxrplc y
y =. ('y';'{:') rxrplc y
NB. y =. ('y';'(1&{)') rxrplc y
```

```
'(', y, ')'
)

NB. (d '*:@y - *:@x') dfunc 2 3 => 2 3 5(=3^2 - 2^2)
dfunc =: 1 : 0
'X Y' =. y
Z =. ". u ,' ','"(1) ', (":X),' ',(":Y)
X, Y, Z
)
```

```
NB. particular X, Y - solution  ========================
adjtest0 =: 3 : 0
, roundint (200, 200) +"(1) (50, _50) *"(1) y
)

test0 =: 3 : 0
DA =: ''
DB =: ''
j =. 0
while. j < (#INIT)
  do.
    DA1 =. 0.2 (df Eq) rk^:(i. 20) (> j{INIT)
    DA0 =. adjtest0 DA1
    DA =: DA, < DA0
    DB1 =. _0.2 (df Eq) rk^:(i.20) (> j{INIT)
    DB0 =. adjtest0 DB1
    DB =: DB, < DB0
    j =. j + 1
  end.
)
NB. EQ =: ({. - {:)
eq =: df Eq
```

```
NB. direction field calculation  ====================================
NB. point range X = _12 to 12, y = _12 to 12
Dtest2 =:  > , (<"(1)) (i: 9) ,."(0, 1) (i: 9)

NB. adjust to graphic form
adjtest2 =: 3 : 0
, roundint (200, 200) +"(1) (2, _2)* "(1) y
)

NB. make arrow figures from x, y, dy/dx ==========
NB. test2 Dtest => TT
test2 =: 3 : 0
NB. calculate dy/dx
Stest2A =: 0.1 * 2 {"(1) exfunc Dtest2
Dtest =: Dtest2, "(1, 0) Stest2A

TT =: ''
i =. 0
while. i < #Dtest
  do.
    TT =: TT,  < adjtest2 10 * |: > 0.5 arrow i{Dtest
    i =. i + 1
  end.
)

NB. geneate gradient to dif. equation ====================================
wr =: 1!:2&2
exfunc =: 3 : 0
AA =. y
'M N' =. $ AA
eqq =: df Eq
RR =. ''
i =. 0
while. i < ({. $ AA)
  do.
     AAA =.'. (': eqq), ':(i{ AA)
```

```
    RR =. RR , (i{AA), AAA
    i =. i + 1
  end.
(M, (>: N))$ RR
)

NB. Arrow Symbol
NB.  length arrow center slope
NB.   eg. 1 arrow _2 1 0.5
arrow =: 3 : 0
:
require 'plot'
d =. x
'x0 y0 a' =: y
NB. arrow figure
'p0x p0y' =: (-d), 0
'p1x p1y' =. d, 0
'p2x p2y' =. (0.9*d), (0.1*d)
'p3x p3y' =. (0.9*d), (_0.1*d)
'p4x p4y' =. d, 0
NB. plot (_4, 4, 4, _4, _4, p0x, p1x, p2x);(_4, _4, 4, 4, _4, p0y, p1y, p2y)
px =. p0x, p1x, p2x, p3x, p4x
py =. p0y, p1y, p2y, p3y, p4y
NB. rotate arrow
Cos =. 1 % %: 1 + *: a
Sin =. a % %: 1 + *: a
'xx yy' =: (2 2$Cos, (-Sin), Sin, Cos) (+/ . *) (px ,: py)
xx =. x0 + xx
yy =. y0 + yy
NB. plot (_4, 4, 4, _4, _4, xx);(_4, _4, 4, 4, _4, yy)
xx;yy
)
```