Rによる種々の方程式の解法

Solving nonlinear simultaneous equations by R langauge

作花　一志（京都情報大学院大学）
Kazushi Sakka（The Kyoto College of Graduate Studies for Informatics）

Abstract

R 言語は統計処理，データ解析のみならず数値計算に非常に便利である。この小文ではグラフの描き方から始め，線形・非線形方程式の解き方，陰関数で与えられた連立方程式，実解をもたない方程式さらに 1 階微分方程式までのグラフィカルに解く例を示す。

This paper shows the methods and examples of non-linear equations,spontaneous equations, 1st order differential equations.

まずグラフ描画

方程式を解くという学習は中学校以来学んでいるが，公式によって解ける方程式は 1 次方程式と 2 次方程式くらいなもので，きわめて例外である。f(x)=0 を解くとは $x=f^{1}(0)$ を求めること，すなわち y=f(x) と x 軸との交点を求めることである。

そこでまずはグラフを描いてみよう。y=sin(x) を $-5 \leqq x \leqq 5$ で描く最も簡単な方法は google で y = sin(x)　を検索することであるが R プログラミングのコードは

curve(sin(x),-5,5)

だけでよく，一般に関数 f(x) を定義してグラフを描くには

f= function(x) sin(x)
curve(f(x),-5,5)

とすればよい。次の第 1 行は関数定義である。

```
f= function(x) sin(x)
g= function(x) cos(x)
curve(f(x),-5,5)
curve(g(x),-5,5,col=2,add=T)
abline(v=0,col=8)
abline(h=0,col=8)
```

= の代わりに <- を使うことも多い。

またここに別の関数 cos（x）のグラフを重ねて描くには add=T を付け加えるが，これは重ね描きを表す重要な命令で，付けないと前に描いたグラフは消えてしまう。col = の後の数はカラーコードで 1（黒：通常省略），2（赤），3（緑），4（青），5（シアン），6（マゼンタ），7（黄），8（グレー）である。

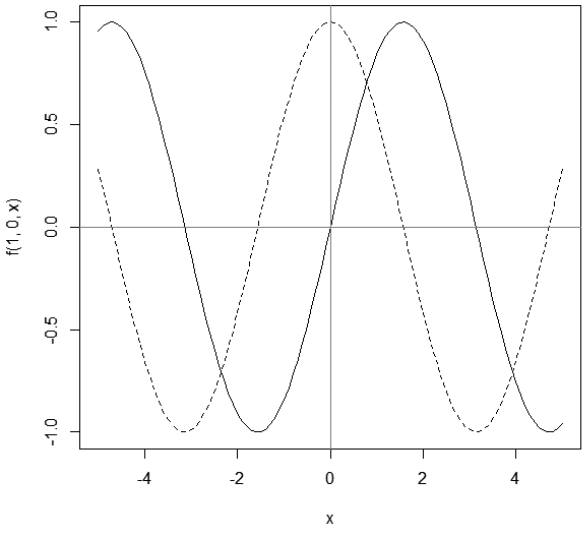

図 1　sin(x) 実線と cos(x) 破線

その他の色は“pink”，“skyblue”のようにすればよい。線種を lty= で表す，1（実線）2（破線）3（点線）。また lwd= で線の幅を指定する。数値が大きいほど太い。なおパラメータ a,b を含めて

f=function (a,b,x) a*sin(x)+b*cos(x) と定義して
curve(f(1,0,x),-5,5)
で sin(x) を
curve(f(0,1,x),-5,5，lty=2, add=T)
で cos(x) を描画してもよい。

陰関数の場合

関数を y=f(x) の形で与えられる場合を陽関数，f(x,y)=0 の形を陰関数という f(x,y)=0 より y=f(x) の形に変換することはしばしば困難である。例えば $x^2+y^2=4$ は容易に $y = \pm\sqrt{(4-x^2)}$ と変更できるが 2 個の関数が必要であるし 3 次の項が含まれると大変だ。そこで陰関数の形のままグラフ表示してみよう。それには等高線を描く関数 contour を使う。$x^3-12xy-y^3+28=0$（f1=0）と $x^2-xy+y^2-12=0$（f2=0 楕円）を描くコードと結果グラフ図 2 を示す。

ここで f2 として単に y とすると f2=0 のグラフは y 軸と重なる

```
######### 陰関数グラフ #############
x =y=seq(-10, 10, 0.2)
 f1=function(x,y) x^3-12*x*y -y^3+28
 f2=function(x,y) x^2-x*y +y^2-12
 z1 = outer(x, y, f1)
 z2 = outer(x, y, f2)
 contour(x, y, z1, levels=0,col=2)
 contour(x, y, z2, levels=0,col=4,add=T)
 abline(h=0,col=8); abline(v=0,col=8)
grid()
text(-5,9,"x^3-12*x*y -y^3+28=0",col=2)
text(5,9,"x^2-x*y +y^2-12=0",col=4)
```

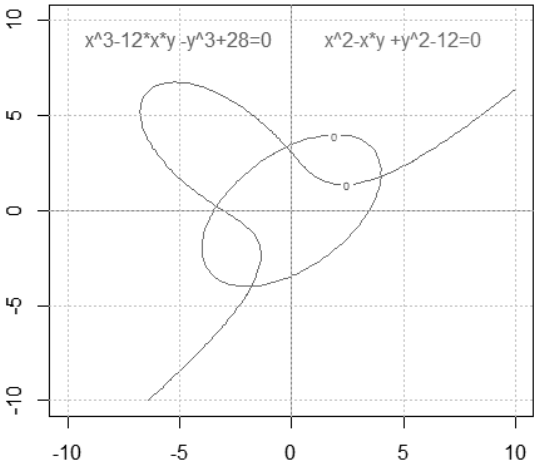

図 2　陰関数表示

極座標の場合

グラフを表す式は絶対値 r と偏角 t を用いて r=f(t) と与えられ，閉曲線を描くとき便利である。

r=2*sin(2*t) の時

x=rcos(t)　y=rsin(t) をプロットするには

```
t=seq(0,2*pi,by=0.05)
 r=2*sin(2*t)
 x=r*cos(t);y=r*sin(t)
plot(x,y,xlim=c(-2,2),ylim=c(-2,2),type="l",xlab="x",ylab="y")
```

これは正葉曲線といわれる。

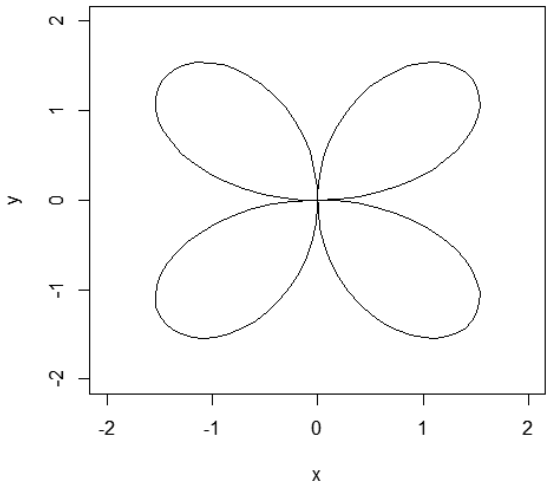

図 3　正葉曲線

方程式解法

前述のとおり方程式 f(x)=0 を解くとは逆変換 $x=f^{-1}(0)$ を求めること，すなわち y=f(x) と x 軸との交点を求めることである

では実際に方程式を解いてみよう。

線形でも非線形でも同じことだから，いきなり非線形方程式　$x^2 - 2^x=0$ を解いてみよう。2^x を y と置き換えて・・・などとしてもうまくいかない。置換が悪いのではなく，そもそもいわゆる数学的解法が無理なのだ。受験技術に長けた大学生よりむしろ直観力のある小中学生のほうが早く解くことができるだろう。x=2 としてみると $2^2 - 2^2=0$ だから解で

```
z=locator(2)                  # グラフと x 軸との交点を挟む 2 点をクリック
  points(z,col=4)             # クリックした 2 点が表示される
  p1=z$x[1] ； p2=z$x[2]      # クリックした 2 点の x 座標
ans=uniroot(f,c(p1,p2))       # その間の解
```

あることはすぐにわかり，また x=4 も明らかに解である。しかし解はこの 2 つだけだろうか。これ以上は直観でも解析でもわからない。

グラフを描いてみるともう一つ負の解があるが，これは数値的にしか求まらない。その解法は昔から多数開発されているが，R では uniroot という便利な関数が装備されている。x 軸との交点の左右に適当な数 x1 と x2 を取り

ans = uniroot(f, c(x1, x2)) とするだけでよい。

解は ans$root に収まっていて，数値 -0.7666825 はコンソールとグラフィック画面に表示される。なお四捨五入して小数点第 3 位まで表すなら round(ans$root,3) とする。

```
# 方程式 f(x)=0　の f(x) を与える
f = function(x) x^2-2^x
curve(f(x),-6,6)　　# この範囲の f(x) プロット
abline(h = 0,  col = 8)　#  x 軸を描く
abline(v = 0,  col = 8) #  y  軸を描く
x1=　 ;x2=　　# x1,x2 の値を与える
ans = uniroot(f, c(x1, x2))　# x1,x2 の間の解
ans$root　　#　解はコンソールに
text (-2,5,ans$root)　解はグラフィック画面に
title(main="Solution  x^2=2^x")
```

でも x1,x2 にどんな値を与えたらいいか？グラフを描いてみてから x1=-2，x2=0 と x 軸との交点の左右の点を決めてもいいが，もっと素晴らしい方法がある。locator を使えばグラフと x 軸との交点を挟む 2 点をクリックするとその 2 点の x 座標が x1,x2 となるのだ。

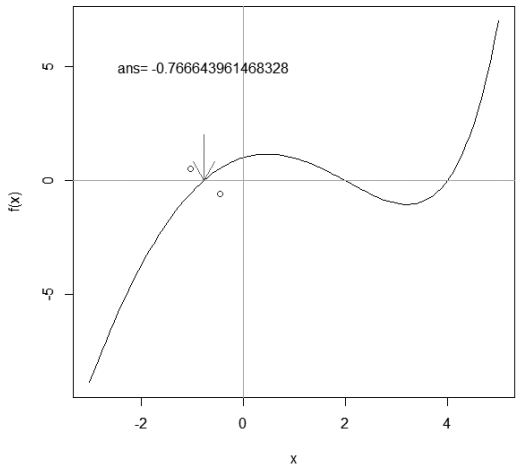

図 4　$x^2=2^x$ の解

これにより関数を変更すればどんな方程式でも微分の知識なし解ける。ただしもちろん実根だけ。1 点クリックで解く方法は後述する。

整方程式の場合は係数をベクトルで与えて polyroot 関数を使えばよい

$x^2+2x-3=0$ の解は

　polyroot(c(-3,2,1))　より求まる。ただし（ ）内は昇べき順の係数である。

解は 1+0i　-3+0i　と求まるが

　　これは x=1，-3 を表す。

また $2x^3+x^2-2x+3=0$　の解は

polyroot(c(3,-2,1,2))　より

 0.5767283+0.7580072i,

 -1.6534566+0.0000000i ，0.5767283-0.7580072i　と求まるが 2 番目の値は実数で　他は複素数（共役）である。

陰関数の場合は陽関数に変換せずに連立方程式の形で解を求めることができる。コードがやや長くなるがマウスクリックするのは 1 回だけでよい

$x^2-3x-3-y=0$

xsin(-x)+2-y=0

の解を求めよう

まず nleqslv というライブラリをインストールしておく。プログラムの第 9 行までは 2 つの曲線のと意義と描画を表している。次に関数 g を第 12 行～第 18 行のように定義して方程式を解く準備をする。

```
x =seq(-6, 6, 0.2);y=seq(-6, 25, 0.2)
f1=function(x,y) x^2-3*x-3-y
f2=function(x,y) x*sin(-x)+2-y
z1 = outer(x, y, f1)
z2 = outer(x, y, f2)
contour(x, y, z1, levels=0,col=2)
contour(x, y, z2, levels=0,col=4,add=T)
abline(h=0,col=8); abline(v=0,col=8)
grid()
#define eqs
library(nleqslv)
g= function(z) {
 x = z[1]
 y = z[2]
 f1 = x^2-3*x-3-y
 f2 =  x*sin(-x)+2-y
```

```
 c(f1, f2)
}
#solve
 p=locator(1); points(p,col=4)
  x0=p$x ;  y0=p$y
ans = nleqslv(c(x0,y0), g)
ans$x;points(ans$x[1],ans$x[2],pch=20,col=2)
arrows(x0,y0,ans$x[1],ans$x[2],lty=3)
 text(x0,10,paste("x=",round(ans$x[1],3)))
```

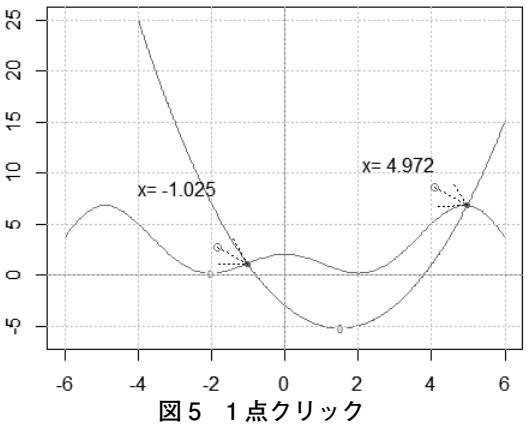

図５　１点クリック

曲線が描かれたグラフ上でクリックした点を(x0,y0) とすれば 2 曲線の交点が画面上に示される。解は ans = nleqslv(c(x0,y0), g) より求まり

交点は x=ans$x[1]　 y=ans$x[2] である。

$f1(x,y)=x^3/2+x^2-x(y+1)-1$

$f2(x,y)=y$

という連立方程式を上記の方法で解いた。f1=0 は x=0 で不連続となり f2=0 は y 軸と重なる

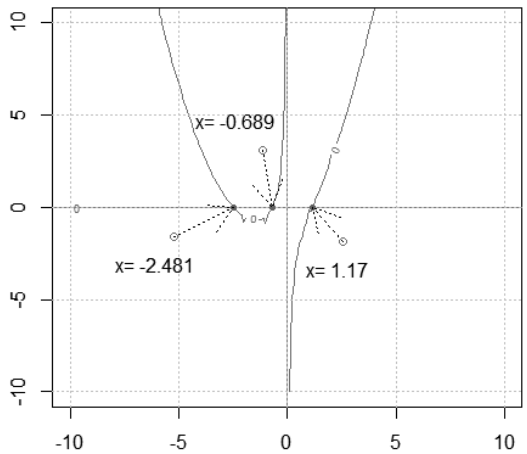

図６　１点クリック

一方 y=0 として 2 個の式をまとめて $x^3/2+x^2-x-1=0$ から前節のように 2 点クリックで求めたものが図 7 である。

当然解は同じである。

以上のことより陰関数で与えられた非線形連立方程式でも実解を 1 点クリックでグラフィカルに求めることができる

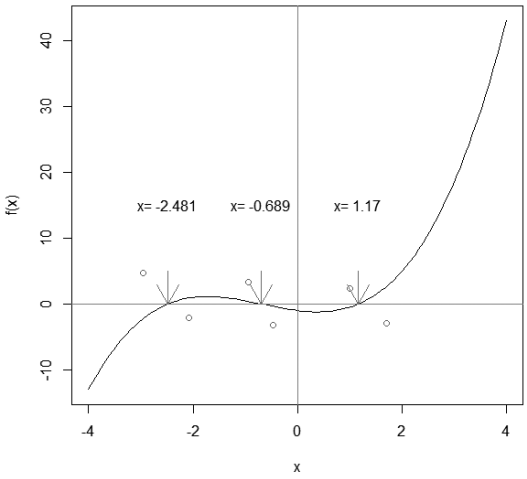

図７　２点クリック

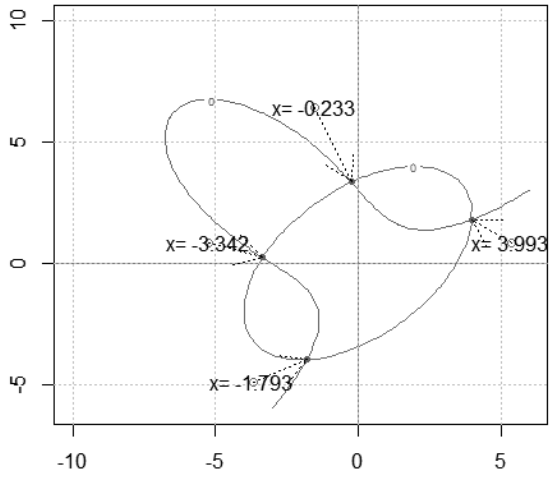

図８　１点クリックによる連立方程式
$x^3-12xy-y^3+28=0$
$x^2-xy+y^2-12=0$
の解は　x=ans$x[1],y=ans$x[2]
曲線は図２と同じ

導関数とニュートン法

関数を expression で定義し D を使うとその導関数が得られる。複雑な公式を使わなくても微分できるので便利である。ただし関数形だけで関数値は求まらないしグラフも描けない

```
y=expression (x^3+a*x+b*cos(x)+log(x)/a)
(Dif1=D(y,"x"))          #y
(Dif2=D(Dif1,"x"))       #y''
(Dif3=D(Dif2,"x") )      #y'''
(Difa=D(y,"a"))          #a で微分
(Difb=D(y,"b"))          #b で微分
```

fn1(x) を微分してその導関数を求めるには
fn1=deriv(~ 関数形，“x”，func=T)
fn2(x) =function(x) attr(fn1(x),"gradient")
とする。
fn2(x)=0 という方程式を解いて，その解に対する fn1(x) の値は極値である。
図 9 は fn1(x)-2sin(x)-x+2 の場合

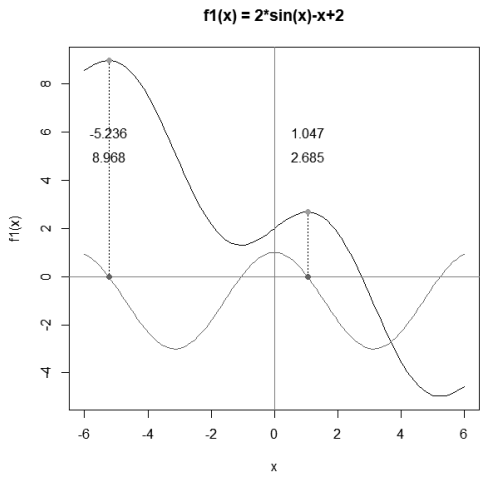

図 9　導関数と極値

ニュートン法で方程式を解く場合には導関数が必要となるが関数形 fn1 を定義すればよい。次のプログラムでは⑥で関数を与え⑦で適当な点をクリックして初期値を与えて解いている。
なお最大反復回数を 100，収束判定定数を 10^{-7} とした。

```
# ニュートン法の関数を定義
newton_method = function(fn1, fn2, x0) {
 eps=1e-7;max=100;x =x0
 for (i in 1:max) {
  dx = fn1(x) / fn2(x)
  x = x - dx
  if (abs(dx) < eps) {
   return(x)
  }
 }
 stop("Not converge")
}
# 方程式とその導関数を定義
 fn1=deriv(~x^2-2^x,"x",func=T)
 fn2=function(x) attr(fn1(x),"gradient")
curve(fn1(x),-2,5)
abline(h=0,col=8) ;abline(v=0,col=8)
# ニュートン法を適用
z=locator(1);points(z,col=4)
x0=z$x
 (ans=newton_method(fn1, fn2, x0))
points(ans,0,col=2,pch=19)
title(paste("x=",round(ans,3)))
```

このプログラムは多少修正すれば実解を持たない方程式にも適応出来る。
例として x − log(x)-1=0 の場合，交点は存在しないがマウスクリックした点 z の x 座標，y 座標を実部虚部に持つ複素数をニュートン法の初期値とすればよい

```
x0=complex(re=z$x,imag=z$y)
```

これによって z の y 座標が正なら複素解
-0.3951+1.7882i　が求まり，負ならその共役の複素解が求まる。同様に x2-x+2=0 の解は 0.5 ± 1.3229i
であり，この値は当然 2 次方程式の解の公式から得られるものと一致する（図 10，図 11）。

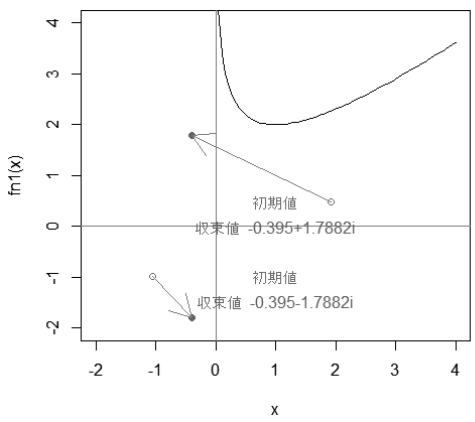

図 10　x-log (8x) +1=0　の複素解

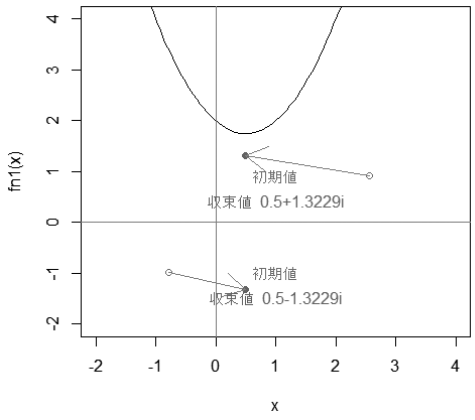

図 11　x^2-x+2=0　の複素解

一階微分方程式

微分方程式を解くとは x，y，y’，y” などを含む方程式から y を x の関数として表すことで，微分積分学修得の後で学ぶのが通例である。その起源はニュートンの運動方程式に始まり，これまでさまざまな解法が研究されているが，解析的方法は一般に非常に難解・技巧的であり，数値的にしか解けない場合も多い。

ところが R には ode と言う便利な関数が装備されていて，y’ と初期値を与えれば，

非線形でも連立でも高階微分の場合でも容易に解くことができる。

例として非線形方程式 y’=x-y/2+2^x*sin(y) を解いてみよう。まず dfn で微分方程式 y’ を定義する。times では 0 から 50 まで 0.1 刻みで y，y’ の値を初期値は x=0，y=1 とする。計算し out という matrix に納める。out の第 1 列，第 2 列は t と y で最後の 10 行 (tail) だけをコンソールに出力した。

また plot によりグラフを描いた（図 12)。y の値はπに収束することがわかる。

なおライブラリ deSolve が必要である。

```
     time       1
[492,] 49.1  3.141593
[493,] 49.2  3.141593
[494,] 49.3  3.141592
[495,] 49.4  3.141593
[496,] 49.5  3.141593
[497,] 49.6  3.141593
[498,] 49.7  3.141593
[499,] 49.8  3.141593
[500,] 49.9  3.141593
[501,] 50.0  3.141593
```

```
library(deSolve)
#   y'=x-y/2+2^(x)*sin(y)
y0=1;fm="x-y/2+2^(x)*sin(y)"
  dfn=function(x,y,parms)
{list(x-y/2+2^(x)*sin(y))}
times = seq(from = 0, to = 50, by = 0.1)
out = ode(y = y0, times = times,
func = dfn,parms=null)
tail (out, n = 10)
plot(out[,1],out[,2],col=2,type="l",xlab="t",ylab="y")
title(paste("y'= ",fm,"    y(0)=",y0))
```

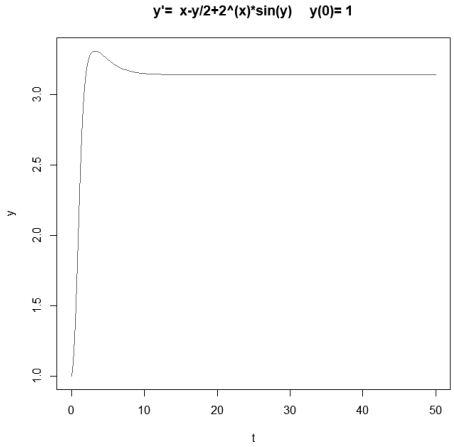

図 12　微分方程式の解　初期値は（0,1）

この結果は初期値 y0=1 という特別な場合果なんだろうか？異なった初期値からでも同じ値が求まるのか？異なった結果になるのか？それを調べるため上記プログラムを一部変更して初期値は画面上を増すクリックして得ることを試みた。その結果初期値 y0>0 の場合はπに y0<0 の場合は - πに収束することが解った。図 13

このように微分方程式を短いコードで簡単に解くことができる。

```
library(deSolve)
plot(0,0,ylim=c(-3,3),type="n");abline(h=0,col=8)
z=locator(1)
x0=z$x;y0=z$y
fm="x-y/2+2^(x)*sin(y)"
  dfn=function(x,y,parms)
{list(x-y/2+2^(x)*sin(y))}
times = seq(from = x0, to = 50, by = 0.1)
out = ode(y = y0, times = times, func = dfn,parms=null)
tail (out, n = 10)
plot(out[,1],out[,2],col=2,type="l",xlab="t",ylab="y")
points(z,col=4,pch=20)
title(paste("y'= ",fm,"  ",out[nrow(out),2]))
```

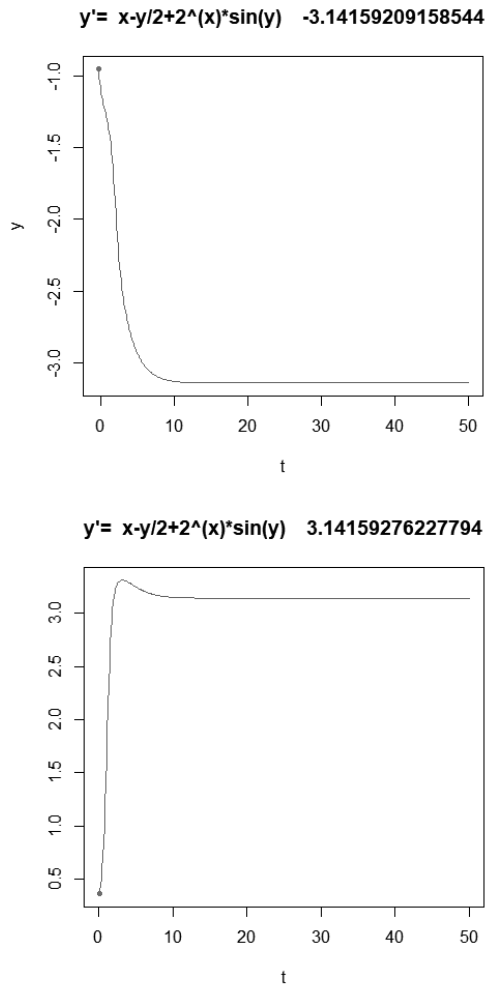

図 13　微分方程式の解　初期値はマウスクリックで指定

参考文献

https://www.f.waseda.jp/sakas/R/Rgraphics17.html
作花　NaisJ. Vol.15 p.74 2021
作花　NaisJ. Vol.17 p.41 2023

◆著者紹介

作花一志　Kazushi Sakka

京都情報大学院大学教授
京都大学理学研究科修了　理学博士
元京都コンピュータ学院鴨川校校長
元国際日本文化研究センター研究員
元日本天文教育普及研究会編集委員長
元京都大学総合人間学部非常勤講師
日本応用情報学会理事
暦学会理事