技術論文

フリーソフトウェアScilab/Scicosによる数値計算

Numerical Computations with Scilab/Scicos

矢田部 学*
Manabu Yatabe

数理モデルを開発する場合、その妥当性を数値計算により確認することが重要である。この際、数値計算に特化したツールを用いると効率的である。モデル検証のために、我々のグループは、多数の数学関数を備えているフリーソフトウェアのScliab/Scicosを使用している。本稿では、Scilab/Scicosによる２つの基本的な数値計算例について紹介する。

In developing mathematical models, it is essential to confirm the validity of the formulations by numerical computations. Use of specialized software is effective for this purpose. To verify the mathematical models, members of our group have employed Scilab/Scicos, a free scientific software package having numerous mathematical functions. In this report, the author shows two basic numerical computations using Scilab/Scicos.

1. まえがき

鎌倉事業部第一技術部では防衛関係のM&S（Modeling and Simulation）業務を行っている。M&S業務においては、基礎原理から新たにモデルを定式化したり、既存のモデルを考え直す作業が発生する。その場合、モデルの妥当性を数値計算により検証することが重要である。この目的のためには、豊富な数学関数やtoolboxを備えたMATLAB※1などの数値計算に特化したソフトウェアの利用が便利である。

しかし、複数の計算機を用いてグループで作業を行う場合、ライセンスの問題が発生する。このような場合には、フリーのソフトウェアの使用が考えられる。そのため、我々のグループではモデルの検証ツールとしてフリーソフトウェアのScilab/Scicosを用いている。

本稿では、数理モデルの検証に有用なScilab/Scicosによる数値計算について紹介する。

2. Scilab/Scicosとは

Scilabは、フランスのINRIA（Institut national de recherche en informatique et en automatique、国立情報学自動制御研究所）とENPC（École nationale des ponts et chaussées、国立土木学校）により開発された数値計算用のソフトウェアである。これは、線形代数、制御系、信号処理、グラフィック機能などに対応する命令や関数が豊富に用意されており、比較的簡単に数値計算プログラムを作成することが可能である。

また、ScicosはScilabに備えられているGUIを有するtoolboxで、この機能を用いるとブロック線図を用いて直截的にモデルを作りシミュレーションを行うことができる。これはMATLABのSimulink※1に類似している。

Scilab/ScicosはWindows、LinuxおよびMacOSに対応し、インターネット上からダウンロードして自由に使用することできる（http://www.scilab.org/）。

使用方法についてはWeb検索、一般の書籍（例えば(1)(2)(3)(4)）、およびScilabのHELP機能などから必要な情報を入手できる。

なお、2009年12月リリースのScilab5.2からScicosは、それを発展させたXcosに変更されている（http://www.scilab.org/products/xcos）。ただし、2010年７月時点では、古いバージョンのScilab/Scicosもダウンロード可能である。

3. 数値計算例

Scilabによる数値計算のプログラムとScicosのブロック線図によるシミュレーションの例を紹介する。ここでは、2007年10月にリリースされたScilab 4.1.2 およびScicos 4.2 を用いた。

※１　MATLAB、Simulinkは、米国The MathWorks社の登録商標である。

＊鎌倉事業部 第一技術部

3.1　Scilabによるプログラミング

図１のような弾性定数kのバネ、減衰定数cのダンパー、および質量mの質点よりなる系を考える。この系には、変位に比例する力$-kx$と速度に比例する力$-c\dot{x}$が作用するので、運動方程式は

$$\ddot{x} = -\omega_0^2 x - 2\gamma\dot{x} \qquad (1)$$

と表される。ただし、$2\gamma \equiv c/m$および$\omega_0^2 \equiv k/m$とおいた。また、ドットは時間微分を表す。

この方程式の解を数値計算により求める。Scilabには、常微分方程式を解く関数odeが用意されているが、プログラム例を示すという目的から、ここでは敢えて４次のRunge-Kutta法[4][5]によるプログラムを作成する。

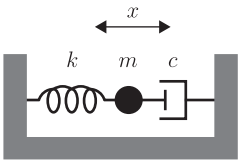

図１　減衰振動モデル

式(1)は、$\dot{x} \equiv y$とおくことにより、１階の連立微分方程式

$$\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\gamma \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \qquad (2)$$

となり、Runge-Kutta法が適用できる。この解法に従ったScilabのプログラムを図２に示す。ここで、Runge-Kutta法のメイン処理が関数rk4であり、微分方程式の定義が関数diffEqsである。この例に示すように、Scilabはベクトルや行列を直接取り扱うことができるために、成分ごとの処理をする必要がなくプログラムが簡潔になる。ただし、Scilabでは、関数rk4で使用するユーザー定義の関数diffEqsを命令getfにより、使用前にロードする必要がある（図２参照）。これはMATLABなどとは異なる。

このプログラムを実行した結果を図３に示す。これより、変位（赤）と速度（青）が時間とともに減衰して行く様子が分かる。なお、ここでは次のパラメータを設定した。

関数rk4（Runge-Kutta法の処理）

```
function [ ] = rk4( )

//   ４次のRunge-Kutta法で
//   微分方程式dx/dt=f(x,t)を初期条件(x0,t0)で解く

getf ('diffEqs.sci');   微分方程式を定義した関数のロード

global omg2 gmm;
omg2 = 2.0; // ばね定数/質量
gmm = 0.2; // 減衰定数/質量

x = [0; sqrt(omg2)]; // 初期条件
tmax = 30; // 評価終了時刻
dt = 0.1; // 積分刻み

i = 1;
T(i,1) = 0; // 初期時刻 ⇒ T(1,1)
X(i,1) = x(1);  X(i,2) = x(2); // 初期値 x ⇒ X(1,:)

for t = dt: dt: tmax

 [k1] = dt*diffEqs(t,x);
 [k2] = dt*diffEqs(t+dt/2,x+k1/2);
 [k3] = dt*diffEqs(t+dt/2,x+k2/2);
 [k4] = dt*diffEqs(t+dt,x+k3);
 x = x+(k1+2*k2+2*k3+k4)/6;        Runge-Kutta法の処理

  i = i+1;
  T(i,1) = t;   X(i,1) = x(1);   X(i,2) = x(2); // プロット用データ
end

// 結果のプロット
clf( );
plot2d(T,X);
xset("font size",3);
xtitle("Dampling Oscillation","Time","Displacement & Velocity");

endfunction
```

関数diffEqs（微分方程式の定義）

```
function [xdot] = diffEqs(t,x);

// 減衰振動の微分方程式

A = [ 0 1; -omg2−2*gmm];
xdot = A*x;

endfunction
```

微分方程式

$$\frac{d\mathbf{X}}{dt} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\gamma \end{pmatrix}\mathbf{X}$$

Runge-Kutta法

$$\frac{d\mathbf{X}}{dt} = \mathbf{f}(t, \mathbf{X})$$

$$\Updownarrow$$

$$\mathbf{X}_{n+1} = \mathbf{X}_n + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$$

where

$$\mathbf{k}_1 = \Delta t\,\mathbf{f}(t_n, \mathbf{X}_n)$$
$$\mathbf{k}_2 = \Delta t\,\mathbf{f}(t_n + \Delta t/2, \mathbf{X}_n + \mathbf{k}_1/2)$$
$$\mathbf{k}_3 = \Delta t\,\mathbf{f}(t_n + \Delta t/2, \mathbf{X}_n + \mathbf{k}_2/2)$$
$$\mathbf{k}_4 = \Delta t\,\mathbf{f}(t_n + \Delta t, \mathbf{X}_n + \mathbf{k}_3)$$

図２　Runge−Kutta法のプログラム例

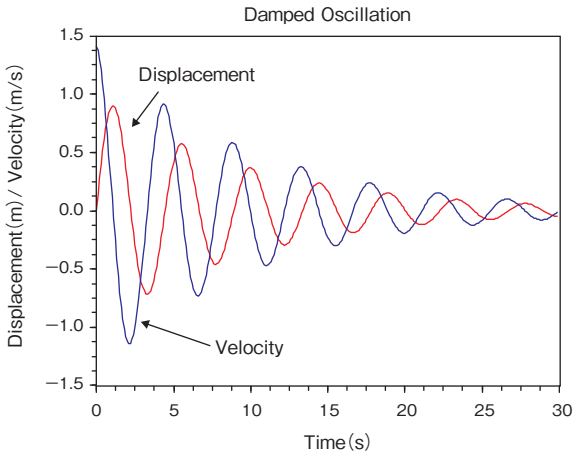

図３　プログラムの実行結果

はね定数/質量：　$\omega_0^2 = 2.0\ [1/\mathrm{s}^2]$
減衰定数/質量：　$\gamma = 0.05\ [1/\mathrm{s}]$
初期変位　　：　$x = 0\ [\mathrm{m}]$
初期速度　　：　$\dot{x} = \sqrt{\omega_0^2 - \gamma^2} \simeq 1.41\ [\mathrm{m/s}]$

3.2　Scicosによるシミュレーション

Scicosのブロック線図を用いたシミュレーションの例として、図１に述べた減衰振動のシステムの変位を測定して、その時点の変位と速度を推定するKalmanフィルターを考える（図４）。ただし、システムはモデル化で考慮されていない擾乱を考慮するためにシステムノイズが加わるものとする。これは測定時の観測ノイズとは異なるものである。

図４の系にKalmanフィルターのアルゴリズムを適用するための定式化について述べる。式(2)より

$$\frac{d\boldsymbol{X}}{dt} = \boldsymbol{A}\boldsymbol{X} \quad \text{ただし} \quad \boldsymbol{X} = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\gamma \end{pmatrix}$$

である。この解は、時刻t_nにおける状態ベクトルを$\boldsymbol{X}_n$、時間刻みを$\Delta t = t_{n+1} - t_n$として、離散的に考えると

$$\boldsymbol{X}_{n+1} = \exp(\Delta t \boldsymbol{A})\boldsymbol{X}_n$$

と書ける。ここで、

$$\exp(\Delta t \boldsymbol{A}) = \boldsymbol{I} + \Delta t \boldsymbol{A} + \frac{1}{2!}(\Delta t \boldsymbol{A})^2 + \cdots \equiv \boldsymbol{F}_n$$

を遷移行列とする。ただし、$\boldsymbol{I}$は２×２単位行列である。このとき、状態方程式は、擾乱（システムノイズ）$\boldsymbol{w}_n$を考慮して

$$\boldsymbol{X}_{n+1} = \boldsymbol{F}_n \boldsymbol{X}_n + \boldsymbol{w}_n \tag{3}$$

と表される。

他方、測定については変位しか観測できないとすると、時刻t_nにおける観測方程式は、観測ノイズを

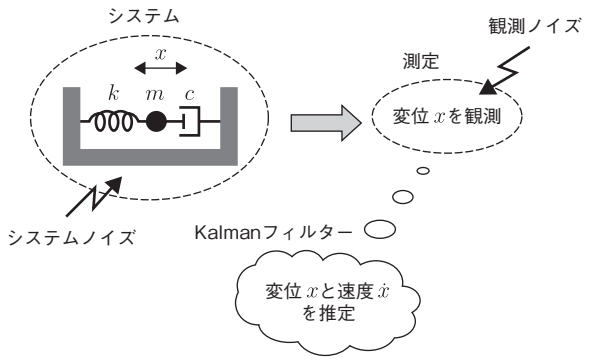

図４　Kalmanフィルター推定のイメージ

$\boldsymbol{v}_n(1 \times 1)$として

$$\boldsymbol{Y}_n = \boldsymbol{H}_n \boldsymbol{X}_n + \boldsymbol{v}_n \tag{4}$$

と書ける。ただし、

$$\boldsymbol{Y}_n = (X_n) \quad (1 \times 1), \qquad \boldsymbol{H}_n = (1 \quad 0)$$

である。いまの場合、$\boldsymbol{Y}_n$と$\boldsymbol{v}_n$はスカラー量であるが、一般性を持たせるためベクトル的にボールド体で表記した。

式(3)と式(4)で表されるシステムについてのKalmanフィルター処理は以下のとおりである（例えば(6)(7)(8)(9)(10)などを参照）。

$$\left.\begin{array}{l} \text{観測更新} \\ \quad \boldsymbol{K}_n = \boldsymbol{P}_n(-)\boldsymbol{H}_n^T[\boldsymbol{H}_n\boldsymbol{P}_n(-)\boldsymbol{H}_n^T + \boldsymbol{R}_n]^{-1} \\ \quad \hat{\boldsymbol{X}}_n(+) = \hat{\boldsymbol{X}}_n(-) + \boldsymbol{K}_n[\boldsymbol{Y}_n - \boldsymbol{H}_n\hat{\boldsymbol{X}}_n(-)] \\ \quad \boldsymbol{P}_n(+) = (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{P}_n(-) \\ \text{時間更新} \\ \quad \hat{\boldsymbol{X}}_{n+1}(-) = \boldsymbol{F}_n\hat{\boldsymbol{X}}_n(+) \\ \quad \boldsymbol{P}_{n+1}(-) = \boldsymbol{F}_n\boldsymbol{P}_n(+)\boldsymbol{F}_n^T + \boldsymbol{Q}_n \end{array}\right\} \tag{5}$$

ここで、式(3)と式(4)で述べた記号以外の新たなものは$\boldsymbol{K}$（Kalmanゲイン）、$\boldsymbol{P}$（推定誤差の共分散行列）、$\boldsymbol{Q}$（システムノイズの共分散行列）、および$\boldsymbol{R}$（観測ノイズの共分散行列）である。上付き添え字Tは転置行列、－１は逆行列を意味する。

また、推定値であることを強調するために状態ベクトルにハットをつけ、状態ベクトルと共分散行列については観測更新前の値を（－）、観測更新後の値を（+）をつけて区別した。

以上に基づいてKalmanフィルターのシミュレーションを行う。このときのScicosのブロック線図を図５に示す。このブロック線図の意味は次のとおりである。左から右に眺めると、先ず、(I)システムの信号が生成（System）され、(II)それを測定（Measurement）して、(III)Kalmanフィルター処理（Kalman Filter）を行う。

ここで、図５の各処理ブロック（System、Measurement、Kalman Filter）はユーザー定義のブロックであり、図６に示すブロック線図をSuper Block機能を用い

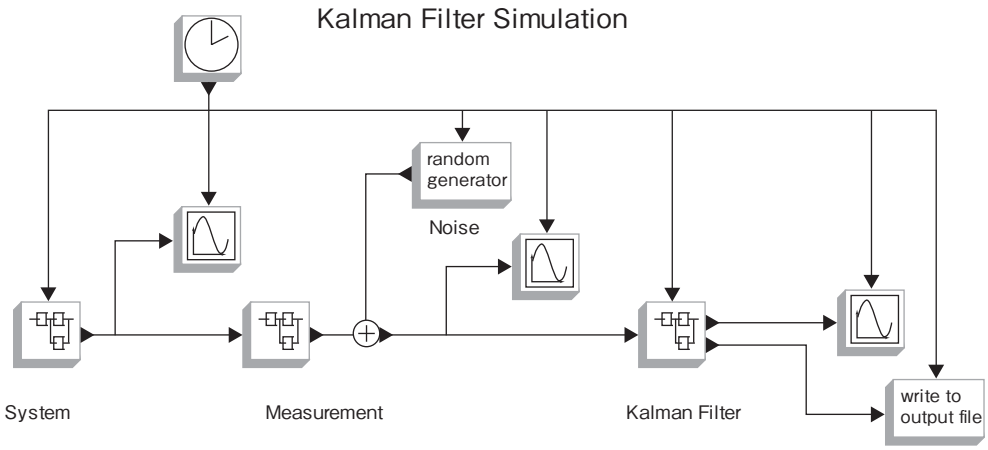

図５　ScicosによるKalmanフィルターシミュレーション

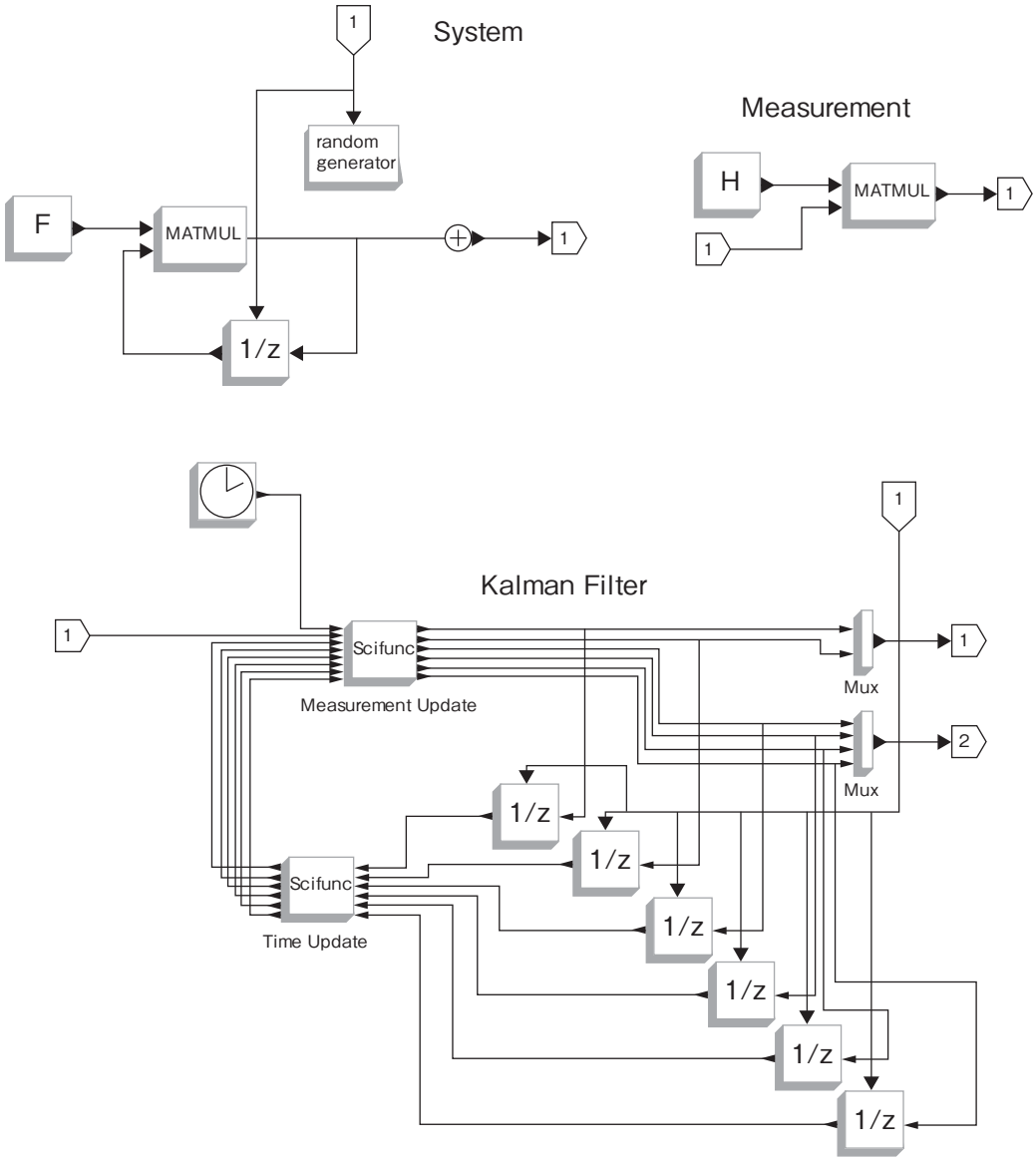

図６　Super Blockの中で定義したブロック線図

てグループ化したものである。図６において、Systemは減衰振動の力学系にシステムノイズが加わった式(3)の処理である。Measurementは式(4)で表される測定装置を模擬する。Kalman Filterは式(5)のKalmanフィルター処理のブロックである。

更に、Kalman Filterでは観測更新（Measurement Update）と時間更新（Time Update）に対してユーザーが任意に処理を定義できる関数Scifuncを用いている。この処理はScilabのスクリプト言語で記述される(図７参照)。これらは式(5)に示した処理である。

図５に示したブロック線図によるシミュレーション結果を図８に示す。図の上段は変位の観測値であり、下段はこの観測値に基づいてKalmanフィルターで推定した変位（赤）と速度（青）である。ここでは、比較のために減衰振動の理論値（システムノイズ無し、黒破線）を重ね描きしている。変位と速度の真の初期値（0, 1.41）から大きく離れたアプオリな初期値（5, 5）から推定を開始したにもかかわらず、Kalmanフィルターが変位と速度を推定していることが分かる。

なお、このシミュレーションで設定したパラメータは以下のとおりである。

Scifunc (Measurement Update)

```
time = u1; Y = u2;
Xm(1) = u3; Xm(2) = u4;
Pm(1,1) = u5; Pm(1,2) = u6; Pm(2,1) = u7; Pm(2,2) = u8;

if modulo(time,dt_obs) <= 1e-5 then
  K = Pm*H'*inv(H*Pm*H'+R);
  Xp = Xm+K*(Y-H*Xm);
  Pp = (eye(2,2)-K*H)*Pm;
else
  Xp = Xm;
  Pp = Pm;
end

y1 = Xp(1); y2 = Xp(2);
y3 = Pp(1,1); y4 = Pp(1,2); y5 = Pp(2,1);  y6 = Pp(2,2);
```

Scifunc (Time Update)

```
Xp(1) = u1; Xp(2) = u2;
Pp(1,1) = u3; Pp(1,2) = u4; Pp(2,1) = u5; Pp(2,2) = u6;

Xm = F*Xp;
Pm = F*Pp*F'+Q;

y1 = Xm(1); y2 = Xm(2);
y3 = Pm(1,1); y4 = Pm(1,2); y5 = Pm(2,1); y6 = Pm(2,2);
```

図７　Scifuncの記述例

システム

ばね定数/質量	:	$\omega_0^2 = 2.0\ [1/\mathrm{s}^2]$
減衰定数/質量	:	$\gamma = 0.05\ [1/\mathrm{s}]$
初期変位	:	$x = 0\ [\mathrm{m}]$
初期速度	:	$\dot{x} = \sqrt{\omega_0^2 - \gamma^2} \simeq 1.41\ [\mathrm{m/s}]$
システムノイズ	:	$N(0, 0.05^2)$ に従う正規乱数

測定

観測間隔	:	$\Delta t = 1.0\ [\mathrm{s}]$
観測ノイズ	:	$N(0, 0.2^2)$ に従う正規乱数

Kalmanフィルター

状態ベクトルの初期値	:	$\hat{\boldsymbol{X}}_0 = (5 \quad 5)^T$
誤差共分散行列の初期値	:	$\boldsymbol{P}_0 = 0.5^2\boldsymbol{I}(2 \times 2)$
システムノイズの共分散行列	:	$\boldsymbol{Q} = 0.05^2\boldsymbol{I}(2 \times 2)$
観測ノイズの共分散行列	:	$\boldsymbol{R} = 0.2^2\boldsymbol{I}(1 \times 1)$

4. むすび

本稿では、M&S業務で数理モデルの検証に利用しているフリーソフトウェアのScilab/Scicosの数値計算例について紹介した。

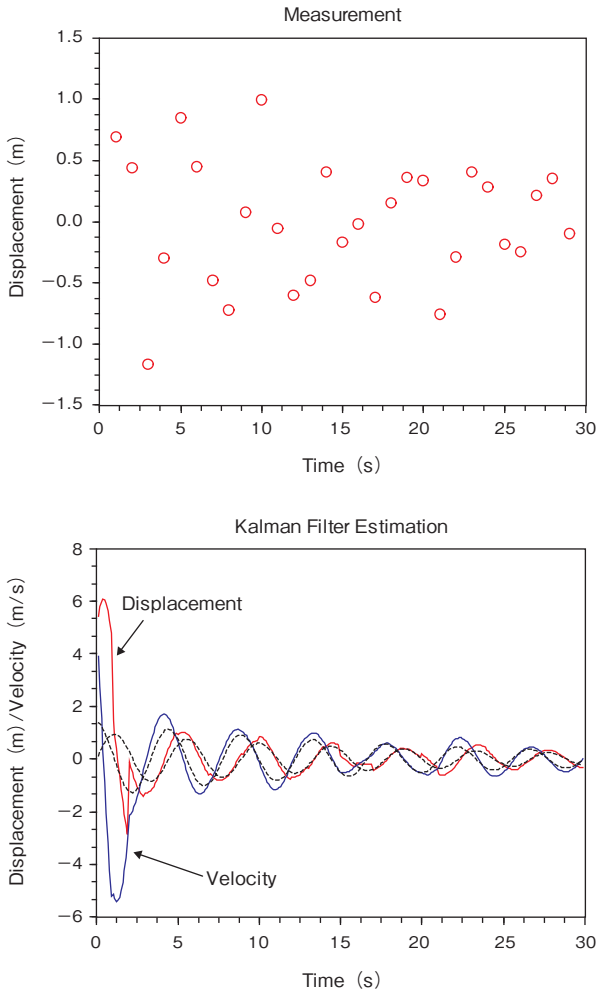

図８　Kalmanフィルターシミュレーションの実行結果

これはフリーソフトウェアなので業務から離れて自己啓発のためにも使える。例えば、物理や工学などの理論を修得する際、身近な数値計算ツールとして利用すれば理解が深まる。興味を持たれた方は、Scilab/Scicosを試みられることをお勧めする。

参考文献

(1) 櫻井鉄也：MATLAB/Scilabで理解する数値計算、東京大学出版会、2003
(2) Campbell, S., J. Chancelier and R. Nikoukhah：Modeling and Simulation in Scilab/Scicos, Springer, 2006
(3) 橋本洋志、石井千春：Scilab/Scicosで学ぶシミュレーションの基礎－自然・社会現象から、経済・金融、システム制御まで，オーム社、2008
(4) 川田昌克：Scilabで学ぶわかりやすい数値計算法、森北出版、2008
(5) Woan, G.：The Cambridge Handbook of Physics Formulas, Cambridge Univ. Press, 2000
(6) Gelb, A.：Applied Optimal Estimation, MIT Press, 1974
(7) 片山　徹：応用カルマンフィルタ、朝倉書店、1983
(8) Brown, R. and P. Hwang：Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises and Solutions, John Wiley & Sons, 1996
(9) Grewal, M. and A. Andrews：Kalman Filtering: Theory and Practice Using MATLAB, John Wiley & Sons, 2001
(10) 西山　清：最適フィルタリング、培風館、2001