Exercises in Scilab

Contents

1 Introduction to Scilab

Remarks

1. All variables are matrices. Scalar is matrix a(1,1). Vector in first row is a(1,:), in first column a(:,1). The sign : means "all".
2. Semi-column ; means: no response. If there is comma or nothing in the end of a command, its value is printed on the screen.
   Remark: the command mode(0) must be called at the beginning.
3. help „object" gives help on "object".
   Icon **?** calls the main help.
4. Comment begins with //.

Variables and operations

There are the following main typed of variables:

- **čísla (matice)**
  Definition:
  - scalar a=5;
  - row vector a=[3 5 1];
  - column vector a=[3; 5; 1], which is the same as a=[3 5 1]'
  - matrix a=[2 3 4; 8 7 6];
  - command a=5:8 creates the vector [5 6 7 8]; 5:2:13 = [5 7 9 11 13]
  - command a=zeros(2,3) creates matrix 2×3 from zeros
  - command a=ones(2,3) creates matrix 2×3 from ones
  - transposition is performed by ' (apostroph)
  - matrix b (3×3) can be composed like this: b=[a; 2*a; 5*a];
  Operations:
  - product of matrices * division / power ^ or ** square root sqrt()
  - dot operations .* ./ .^ are performed entry by entry
  - in operation * the rules of matrix product hold
  - operation a/b means multiplication of a by inversion of b
  (inversion itself is inv(b))
- **text**: a='hello'. It is a vector of letters can be concatinated:
  a='hello '; b='boys' a c=a+b, then c='hello boys'.
  Conversion: s=string(a) gives value of variable a as a string
- **logical variables** - their valies are „true" (=1) a „false" (=0).
  Logical operations: == ~= < <= > >= & (and) | (or) ~ (not)

Examples

Set:

a=[1 2 3] b=[8; 9] c=[11 12 13; 21 22 23; 31 32 33];

Try and justify:

x1=a*a' x2=a'*a y=[[a;5*a] b] c(2,:).*a c(1,2:3)*b

c(3,:).^c(1,:) c(3,:)**2 d1=c(:) dd=c'; d2=dd(:) d2(3:2:7)

Set:

u='first' v='attempt' x=%t (setting of "true") y=5==5 z=5>5

Try and justify:

u+' '+v x & y x & z x | y x | z

Work with variables

- Command who_user(); gives information about defined variables.
- [m,n]=size(a), m=size(a,1), n=size(a,2) give dimensions of the matrix a, resp. number of rows, number of columns. Instead of 1 a 2 one can use 'r' a 'c'.
- n=length(a) number of elements of a.
- n=max(size(a)) length of a vector
- clc clears screen
- clear clears variables
- xdel(winsid()) clears all graphs (close clears the last one)

Programming commands

- **Condition if**

```
if b>c,
    a=5;
else
    a=0;
end
```

  If b>c is true, it is preformed a=5; otherwise a=0;.

  Example

```
// Determine c as bigger from  a, b
a=rand(1,1,'n'); b=rand(1,1,'n');
if a>b, c=a;
else c=b;
end
printf('a = %g, b = %g, c = %g\n',a,b,c)
```

- **Branching of program**

```
select i,
    case 1, prikaz_A;
    case 2, prikaz_B;
    else prikaz_D
end
```

According to i the respective command is performed.

Example
```
// According to i perform
// set the vectors
a=[1 3 5]; b=[2 4 6];
// 1 - addition
// 2 - scalar product
// 3 - tenzor product
// set the operation
```

cont.
```
i=2;
select i
case 1, d=a+b;
case 2, d=a*b';
case 3, d=a'*b;
end
disp(d,'result')
```

- **Cycle for**

```
for i=1:5
    a(i)=2*i;
end
```

For `i=1,2,3,4,5` the command `a(i)=2*i;` is performed. :Result is `a=[2, 4, 6, 8, 10].`

Example 1
```
// Determine weighted sum
x=[1 2 3 4 5 6]; // numbers
p=[.1 .3 .2 .1 .2 .1]; // weights
n=length(x);
s=0;
for i=1:n
s=s+x(i)*p(i);
end
disp(s,'the average is')
```

Example 2
```
// Order numbers according to magnitude
n=10; // how many numbers
a=fix(100*rand(1,n,'u')); // čísla
disp(a,'original numbers')
b=[];
for i=1:n
[x,j]=min(a);
b=[b x];
a(j)=%nan;
end
disp(b,'ordered numbers')
end
```

- **Control of the program**

  pause stops the program.
  resume resumes the program after pause
  abort stops the program definitely.

- **Calling of subprogram**
  exec('my_program',-1) runs the program my_program (-1 suppresses response)

- **Loading functions to memorry**
  getd('my_address') loads all subroutines in the address moje_adresa
  (Scilab does not have path. It knows only the loaded functions)

Printing

Commands disp and fprintf .

- disp(a) shows value of a.
- disp(a,'text') gives value and the text
- printf('entry %d of vector a is %g\n',i,a(i));
  gives e.g.: *entry 5 of vector a is 4.12*

Graphical output

Two-dimensional graph can be constructed by `plot`.

Examples:

- `plot(y)` draws values of `y`.
- `plot(x,y)` draws values of `y` against of `x` (so called xy-graf).
- `plot(a)` draws columns of matrix `a`.

Formatting of a graph:

Line	**Points**	**Color**
- (full)	. (point)	r (red)
: (dotted)	+ (plus)	g (green)
-. (dot dashed)	o (ring)	b (blue)
– (dashed)	x (cross)	w (white)

For more details, call: `help plot` or go to Scilab help: SCILAB HELP >> GRAPHICS > GLOBALPROPERTY

Examples:

- `plot(x,'or')` draws `x` using red crosses.
- `plot(x,y,'r-+',u,v,'b-x')` draws two curves `(x,y)` a `(u,v)`; the first one is red by full line with pluses, the second one by blue line with crosses.

2 Programs for exercises from STS

2.1 Simulation with regression model

```
// T11simCont.sce
// SIMULATION OF THE SECOND ORDER REGRESSION MODEL
// Experiments
// - change parameters of the model
// - change the input signal
// - try to increase the model order to 3
// -------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// PARAMETERS OF THE SIMULATION
nd=100;                         // length of data
a=[.4 .2];                      // parameters at y
b=[1 .2 -.5];                   // parameters at u
k=0;                            // constant (model absolute term)
s=.1;                           // noise variance

y(1)=1; y(2)=3;                 // initial conditions for output
u=sin(10*%pi*(1:nd)'/nd)+.001*rand(nd,1,'n'); // input

// TIME LOOP OF THE SIMULATION
th=[a b k]';                    // vector of parameters
for t=3:nd
  // regression vector
  ps=[y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1]';
  // regression model
  y(t)=th'*ps+s*rand(1,1,'n');
end

// RESULTS OF THE SIMULATION
set(gcf(),"position",[700 100 600 500])
subplot(211),plot(1:nd,u),title('Input')
subplot(212),plot(1:nd,y),title('Output')
```

2.2 Simulation with discrete model

```
// T13simDisc.sce
// SIMULATION OF DISCRETE MODEL
//  (multinomial model - controlled coin with memmory)
//         f( y(t) | u(t),y(t-1) ),  y,u=1,2
// Experiments
// - set the parameters to obtain a deterministic model
// - try to extend the model to f(y(t)|u(t),y(t-1),u(t-1))
//   and values 1,2,3.
// ----------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// PARAMETERS OF THE SIMULATION
// model parameter (conditional probabilities)
// u(t) y(t-1)
//  11  12  21 22
// --------------------------
th= [.2 .6 .9 .3    // y(t)=1
     .8 .4 .1 .7];  // y(t)=2

nd=50;                        // number of steps
u=(.3<rand(1,nd,'u'))+1;      // control variable P(u=1)=.3, P(u=2)=.7
y(1)=1;                       // initial condition for output

// TIME LOOP OF THE SIMULATION
for t=2:nd
  i=2*(u(t)-1)+y(t-1);        // row in the model parameter
  y(t)=sum(cumsum(th(:,i))<rand(1,1,'u'))+1;
                              // generation of the output
end

// RESULTS OF THE SIMULATION
subplot(211),plot(1:nd,u,'g:.')
set(gcf(),"position",[700 100 600 500])
title('Input')
set(gca(),'data_bounds',[0 nd+1 .9 2.1])
subplot(212),plot(1:nd,y,'b:.')
title('Output')
set(gca(),'data_bounds',[0 nd+1 .9 2.1])
```

2.3 Simulation with state-space model

```
// T15simState.sce
// SIMULATION WITH RM IN A STATE-SPACE FORM
// Experiments
// - extend to third order model
//   y(t)=b0.u(t)+a1.y(t-1)+b1.u(t-1)+a2.y(t-2)+b2.u(t-2)+
//        +a3.y(t-3)+b3.u(t-3)+k+e(t)
// -------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')
rand('seed',0)

// PARAMETERS OF THE SIMULATION
nd=100;                       // number of data

et=rand(1,nd,'n');
u=rand(1,nd,'n');             // input
a=[.6 .1]; b0=.8; b=[.3 .2]; k=2; cv=.1; // model parameterers

// REGRESSION REALIZATION
yr(1)=0; yr(2)=1;
for t=3:nd
  er=sqrt(cv)*et(t);
  yr(t)=a*[yr(t-1) yr(t-2)]'+b0*u(t)+b*[u(t-1) u(t-2)]'+k+er;
end

// STATE-SPACE REALIZATION
M=[a(1) b(1) a(2) b(2) k
   0    0    0    0    0
   1    0    0    0    0
   0    1    0    0    0
   0    0    0    0    1];
N=[b0 1 zeros(1,3)]'
A=[1 zeros(1,4)];
B=0;

y(1)=0; y(2)=1;
xt(:,2)=[y(2) u(2) y(1) u(1) 1]';  // initial conditions for state

// time loop of simulation
for t=3:nd
  es=[sqrt(cv)*et(t) zeros(1,4)]';
  xt(:,t)=M*xt(:,t-1)+N*u(t)+es;
  y(t)=A*xt(:,t);
end

// RESULTS OF SIMULATION
scf(1); plot(1:nd,y,1:nd,yr,'.','markersize',4)
legend('state model','regression model');
```

2.4 Least squares estimation

```
// T21estCont_LS.sce
// ESTIMATION OF 2ND ORDER REGRESSION MODEL
//   least squares estimation (off-line)
// Experiments
// - extend the model order
// ----------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// SIMULATION
// parameters
nd=100;                      // length of data
a=[.4 .2];                   // parameters at y
b=[1 .2 -.5];                // parameters at u
k=0;                         // constant (model absolute term)
s=.1;                        // noise variance

y(1)=1; y(2)=3;             // initial conditions for output
u=sin(10*%pi*(1:nd)/nd)'+.001*rand(nd,1,'n'); // input

// time loop
th=[a b k]';                 // vector of parameters
for t=3:nd
  // regression vector
  ps=[y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1]';
  // regression model
  y(t)=th'*ps+s*rand(1,1,'n');
end

// ESTIMATION
for t=3:nd
  Y(t,1)=y(t);
  X(t,:)=[y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1];
end
th=inv(X'*X)*X'*Y;           // regression coefficients
yp=X*th;                     // prediction (for verification)
r=variance(y-yp);            // noise variance

// Results
disp('Parameter estimates')
th,r

set(scf(1),'position',[800 100 600 400]);
plot(1:nd,y,1:nd,yp)  // comparison od output and prediction
legend('optput','prediction');
title('Verification of the estimates')
```

2.5 Estimation with continuous model

```
// T22estCont_B.sce
// ESTIMATION OF 2ND ORDER REGRESSION MODEL
// - Bayesian on-line estimation with statistic update
// Experiments
// - rewrite the program to a single time loop (on-line estimation)
// ---------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// SIMULATION
// parameters
nd=100;                         // length of data
a=[.4 .2];                      // parameters at y
b=[1 .2 -.5];                   // parameters at u
k=0;                            // constant (model absolute term)
s=.1;                           // noise variance

y(1)=1; y(2)=3;               // initial conditions for output
u=sin(10*%pi*(1:nd)/nd)'+.001*rand(nd,1,'n'); // input

// time loop
th=[a b k]';                    // vector of parameters
for t=3:nd
  // regression vector
  ps=[y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1]';
  // regression model
  y(t)=th'*ps+s*rand(1,1,'n');
end

// ESTIMATION
V=1e-8*eye(7,7);                // initial information matrix
for t=3:nd
  psi=[y(t:-1:t-2); u(t:-1:t-2); 1];         // reg. vector
  V=V+psi*psi';                              // updt of information matrix
  Vy=V(1,1); Vyp=V(2:$,1); Vp=V(2:$,2:$);    // divisioning of inf. matrix
  th(:,t)=inv(Vp+1e-8*eye(Vp))*Vyp;          // pt estimates of reg. coefficints
  r(t)=(Vy-Vyp'*inv(Vp+1e-8*eye(Vp))*Vyp)/t; // pt estimates of noise variance
end

// PREDICTION (as evaluation)
thP=th(:,$);                                 // vector of parameters
for t=3:nd
  // regression vector
  ps=[y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1]';
  // regression model
  yp(t)=thP'*ps+sqrt(r($))*rand(1,1,'n');
end

// Results
set(scf(1),'position',[50 300 400 400])
for i=1:6
  subplot(6,1,i),plot(th(i,:))
  if i==1, title('Regression coefficients'), end
```

```
end
set(scf(2),'position',[50 10 400 200])
plot(r)
title('Noise variance')

set(scf(3),'position',[500 60 800 500])
plot(1:nd,y,1:nd,yp)
title 'Output and its prediction'
```

2.6 Estimation with discrete model

```
// T23estDisc.sce
// ESTIMATION OF DISCRETE MODEL
//    f(y(t)|u(t),y(t-1)) with y,u from {0,1}
// Experiments
// - change number of values of individual variables
// - increase the model order
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// SIMULATION
nd=200;                       // number of steps
// parameters of simulation
thS= [.2 .6 .9 .3
      .8 .4 .1 .7];
thU=[.3 .7];
y(1)=1; u(1)=1;               // initial condition for output
// time loop of simulation
for t=2:nd
  u(t)=sum(cumsum(thU)<rand(1,1,'u'))+1;
                              // control variable P(u=1)=.3, P(u=2)=.7
  i=2*(u(t)-1)+y(t-1);        // row in the model parameter
  y(t)=sum(cumsum(thS(:,i))<rand(1,1,'u'))+1;
                              // generation of the output
end

// ESTIMATION
V=zeros(2,4);                 // initial statistics
for t=2:nd
  i=2*(u(t)-1)+y(t-1);        // row of model matrix
  V(y(t),i)=V(y(t),i)+1;      // updt of statistics

  for j=1:4                   // point estimates (normalization)
    tht(:,j)=V(:,j)/sum(V(:,j));
  end
  thE(t,:)=tht(1,:);          // remember
end

// PREDICTION
yp(1)=1;
for t=2:nd
  i=2*(u(t)-1)+y(t-1);        // row in the model parameter
  yp(t)=sum(cumsum(thS(:,i))<rand(1,1,'u'))+1;
end

// Results
set(scf(1),'position',[60 60 600 500])
for i=1:4
  subplot(4,1,i)
  plot(thE(:,i))                                    // estimated
  plot((nd-199:nd),ones(1,200)*thS(1,i),':r')     // true
  set(gca(),'data_bounds',[0 nd -.1 1.1])
  if i==1,
```

```
    title('Evolution ot parameter estimates (left column, only)')
    legend('estimated','true',[350 1.2]);
  end
end

s=nd+1-20:nd;
set(scf(2),'position',[700 60 600 500])
plot(s,y(s),'x:',s,yp(s),'o:','markersize',10)
title 'Prediction'

disp('Simulated parameter',thS)
disp('Estimated parameter',tht)
```

2.7 Prediction with continuous model

```
// T31preCont.sce
// NP-STEP PREDICTION WITH CONTINUOUS MODEL (KNOWN PARAMETERS)
// Experiments
// Change: - np = number of steps of prediction
//         - r  = noise variance
//         - th = model parametrs
//         - u  = input signal
// -------------------------------------------------------------------
exec('SCIHOME/ScIntro.sce',-1), mode(0), getd('functions')

nd=100;                            // number of data
np=5;                              // length of prediction (np>=1)
//  b0 a1 b1  a2 b2 k
th=[1 .4 -.3 -.5 .1 1]';           // regression coefficients
r=.02;                             // noise variance
u=sin(4*%pi*(1:nd)/nd)+rand(1,nd,'n'); // input
y=ones(1,2);

nu=zeros(4,2);
yp=ones(1,2);

// TIME LOOP
for t=3:(nd-np)                   // time loop (on-line tasks)
  // prediction
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]'; // first reg. vec for prediction
  yy=ps'*th;                      // first prediction at t+1
  for j=1:np                       // loop of predictions for t+2,...,t+np
    tj=t+j;                        // future times for prediction
    ps=[u(tj); yy; ps(1:$-3); 1];  // reg.vecs with predicted outputs
    yy=ps'*th;                     // new prediction (partial)
  end
  yp(t+np)=yy;                     // final prediction for time t+np
  // simulation
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]';  // regression vector for sim.
  y(t)=ps'*th+sqrt(r)*rand(1,1,'n');      // output generation

end

// Results
s=(np+2):(nd-np);
scf(1);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[1 nd -3 5])
legend('output','prediction');
title(string(np)+'-steps ahead prediction')
```

2.8 Adaptive prediction with continuous model

2.9 Prediction with discrete model with known parameters

```
// T33preCat.sce
// PREDICTION WITH DISCRETE MODEL
// - known model parameters
// Experiments
// Change: - np  = number of steps of prediction
//         - th1 = model parametrs
//         - u   = input signal (effect on estimation)
//         - uncertainty of the system (effect on estimation)
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=100;                           // length of data sample
np=5;                             // length of prediction (np>=1)
th1=[0.98 0.01 0.04 0.97];        // parameters for simulation (for y=1)
th=[th1; 1-th1];                  // all parameters
u=(rand(1,nd)>.3)+1;              // input
y=1;

// TIME LOOP
for t=2:(nd-np)
  // prediction
  i=2*(u(t)-1)+y(t-1);            // column of model matrix
  yj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  for j=1:np                      // inner loop of prediction
    i=2*(u(t+j)-1)+yj;            // column of model matrix
    yj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // prediction
  end
  yp(t+np)=yj;                    // remember last prediction

// simulation
  i=2*(u(t)-1)+y(t-1);            // column of the table
  y(t)=sum(cumsum(th(:,i))<rand(1,1,'u'))+1;  // output generation
end


// RESULTS
disp(' Model parameters',th), printf('\n')

s=(np+3):(nd-np);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gcf(),'position',[600 100 800 400])
set(gca(),"data_bounds",[0 nd+1 .9 2.1])
legend('output','prediction');
title(string(np)+'-steps ahead prediction')

Wrong=sum(y(s)~=yp(s)), From=nd
```

2.10 Off-line adaptive prediction with discrete model

```
// T34preCat_OffEst.sce
// PREDICTION WITH DISCRETE MODEL (OFF-LINE ESTIMATION)
// Experiments
// Change: - length of prediction
//         - uncertainty of the simulated model
//         - imput signal (effect on estimation)
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=200;                             // number of data
np=3;                               // length of prediction
th1=[0.98 0.01 0.04 0.97];              // parameters for simulation (for y=1)
th=[th1; 1-th1];                    // all parameters
u=(rand(1,nd)>.3)+1;                // input
y=1;

// SIMULATION (measured data)
for t=2:nd
  i=2*(u(t)-1)+y(t-1);              // row of the table
  y(t)=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // output generation
end

// ESTIMATION (on the whole dataset)
nu=zeros(2,4);
for t=2:nd
  i=2*(u(t)-1)+y(t-1);              // row of the table
  nu(y(t),i)=nu(y(t),i)+1;          // statistics update
end
Eth=nu./(ones(2,1)*sum(nu,1));      // estimate of parameters

// PREDICTION
for t=2:(nd-np)
  i=2*(u(t)-1)+y(t-1);              // column of model matrix
  yj=sum(cumsum(Eth(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  for j=1:np                        // inner loop of prediction
    i=2*(u(t+j)-1)+yj;              // column of model matrix
    yj=sum(cumsum(Eth(:,i))<rand(1,1,'u'))+1; // prediction
  end
  yp(t+np)=yj;                      // remember last prediction
end

// RESULTS
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)

s=(np+2):(nd-np);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gcf(),'position',[800 100 500 400])
set(gca(),"data_bounds",[0 nd+1 .9 2.1])
legend('output','prediction');
```

```
title(string(np)+'-steps ahead prediction')

Wrong=sum(y(s)~=yp(s))
From=nd
```

2.11 On-line adaptive prediction with discrete model

```
// T35preCat_OnEst.sce
// PREDICTION WITH DISCRETE MODEL (ON-LINE ESTIMATION)
// Change: - length of prediction
//         - uncertainty of the simulated model
//         - imput signal
//         - study the beginning when estimation is not finished
//           how can we secure quicker transient phase of estimation?
// ----------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=100;                            // number of data
np=5;                              // length of prediction
th1=[0.98 0.01 0.01 0.98];         // parameters (for y=1)
th=[th1; 1-th1];                   // all parameters
u=(rand(1,nd,'u')>.3)+1;           // input
y=1;

// TIME LOOP
Et=zeros(nd-np,4);
nu=1e-8*rand(2,4,'u');
Eth=nu./(ones(2,1)*sum(nu,1));     // pt estimates;
for t=2:nd-np                      // time loop

  // prediction
  i=2*(u(t)-1)+y(t-1);             // column of model matrix
  yj=sum(cumsum(Eth(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  for j=1:np                       // loop of prediction
    tj=t+j;
    i=2*(u(tj)-1)+yj;              // column of model matrix
    yj=sum(cumsum(Eth(:,i))<rand(1,1,'u'))+1; // prediction
  end
  yp(t+np)=yj;                     // remember last prediction

  // simulation
  i=2*(u(t)-1)+y(t-1);
  y(t)=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // output

  // estimation
  i=2*(u(t)-1)+y(t-1);             // row of model matrix
  nu(y(t),i)=nu(y(t),i)+1;         // statistics update
  Eth=nu./(ones(2,1)*sum(nu,1));   // pt estimates
  Et(t,:)=Eth(1,:);
end

// Results
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)
title 'Evolution of parameter estimates'

subplot(121),plot(Et)
```

```
set(gcf(),'position',[500 100 1000 400])
set(gca(),"data_bounds",[0 nd+1 -.1 1.1])
subplot(122)
s=(np+2):(nd-np);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[0 nd+1 .9 2.1])
legend('output','prediction');
title([string(np),'-steps ahead prediction'])

Wrong=sum(y(s)~=yp(s))
From=nd
```

2.12 State estimation

```
// T46statEst_KF.sce
// STATE ESTIMATION (KALMAN FILTER)
// Experiments
// - change model parameters M,N,A,B
// - set different system and model covariances rw,rv and Rw,Rv
// - try lower stat-estimate covariance Rx
// -----------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=200;                // number of data
// SIMULATION
// parameters of simulation
M=[.8 .1
   .3 .6];
N=[.5 -.5]';
A=[.9 -.2];
B=0;
rw=.1*eye(2,2);
rv=.1;
x(:,1)=[0 0]';
u=rand(1:nd,'n');
// time loop of simulation
for t=2:nd
  x(:,t)= M*x(:,t-1)+N*u(t)+rw*rand(2,1,'n');
  y(t)  = A*x(:,t)+B*u(t)+rv*rand(1,1,'n');
end

// ESTIMATION
// initialization of estimation
Rw=1*eye(2,2);                     // state noise covariance
Rv=.1;                             // output noise vovariance
Rx=1000*eye(2,2);                  // estimated state covariance
xp(:,1)=zeros(2,1);                // initial state
// loop for state estimation
for t=2:nd
  [xp(:,t),Rx,yp(t)]=Kalman(xp(:,t-1),y(t),u(t),M,N,[],A,B,[],Rw,Rv,Rx);
end

// RESULTS
subplot(311),plot(1:nd,x(1,:),1:nd,xp(1,:))
set(gcf(),"position",[700 100 600 500])
title('First state entry')
legend('state','estimate');
subplot(312),plot(1:nd,x(2,:),1:nd,xp(2,:))
title('Second state entry')
legend('state','estimate');
subplot(313),plot(1:nd,y,1:nd,yp')
title('Output')
legend('output','estimate');
```

2.13 Noise filtration

```
// T47statEst_Noise.sce
// KALMAN AS A NOISE FILTER
// Experiments
// - change Rw and Rv to catch properly the signal
// - Rw ... changes of the signal
// - Rv ... changes of the noise
// ----------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// SIMULATION
tt=0:.1:(2*%pi);
nt=length(tt);
sd=2;                             // simulation noise
e=[sd*rand(1,nt,'n'); sd*rand(1,nt,'n')];
g=[10*cos(tt); 15*sin(tt)];       // pure signal (ellipse)
x=g+e;                            // measured noisy signal

// FILTRATION
Rz=1e6*eye(2,2);                  // state-estimate cov. matrix
Rw=.01*eye(2,2);                  // state-model cov. matrix
Rv=.1*eye(2,2);                   // output-model cov. matrix
M=[1 0                            // state-model matrices
   0 1];
A=[1 0
   0 1];
N=[0 0]';
F=[0 0]';
B=0;
G=0;
z(:,1)=[0 0]';                    // initial state

for t=2:nt
  [z(:,t),Rz,yp]=Kalman(z(:,t-1),x(:,t),0,M,N,F,A,B,G,Rw,Rv,Rz);
end

// Results
plot(g(1,:),g(2,:),'m:')
plot(x(1,:),x(2,:),'b.')
plot(z(1,:),z(2,:),'r.:')
```

2.14 Control with continuous model

```
// T51ctrlCont.sce
// DYNAMIC PROGRAMMING FOR THE FIRST ORDER RGRESSION MODEL
// Experiments
// Change: - regression coefficients a1,b0 and noise se
//         - penalization of input la
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

a1=.8; b0=1;                  // regression coefficients
se=.01;                       // model noise standard deviation
y0=5;                         // initial condition for output
la=.1;                        // penalization of control variable
nd=30;                        // length of control horizon

// SIMULATION
y=zeros(1,nd); y(1)=y0;
for t=2:nd
  u(t)=0;                     // no control
  y(t)=a1*y(t-1)+b0*u(t)+se*rand(1,1,'n'); // without control
end

// CONTROL OPTIMIZATION
R=0;                          // initial condition for Bellman function
for t=nd:-1:2                 // runs against the time
  L=la/(1+R);
  S(t)=a1*b0/(b0^2+L);        // coefficient at u
  R=L*a1^2/(b0^2+L);          // partial reminder
end

// CONTROL APPLICATION
yr=zeros(1,nd); yr(1)=y0;     // initial condition for output
ur=zeros(1,nd);
for t=2:nd
  ur(t)=-S(t)*yr(t-1);        // optimal control
  yr(t)=a1*yr(t-1)+b0*ur(t)+se*rand(1,1,'n'); // with opt. control
end

// RESULTS
plot(1:nd,y,'b.:',1:nd,yr,'r.:',1:nd,ur,'g.:')
set(gcf(),"position",[700 100 600 500])
title('Control to zero - dynamic programming, 1st order model')
legend('y - without control','yr - with control','ur - control');
```

2.15 Control with discrete model

```
// T52ctrlDisc.sce
// CONTROL WITH DISCTRETE DYNAMIC MODEL
// Experiments
// Change: - criterion om
//         - uncertainty of the system
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

// VARIABLES TO BE SET
nh=30;                        // length of control interval
y0=1;                         // initial condition for output

// y  1  2       u y1  =  criterion
om=[1 2       // 1 1
    2 3       // 1 2
    2 3       // 2 1
    3 4];     // 2 2  ... preference of lower indexes

// y  1  2       u y1  =  system model
th=[.3 .7   // 1 1
    .1 .9   // 1 2
    .4 .6   // 2 1
    .2 .8]; // 2 2   ... rather uncertain system

// computed variables and initializations
fs=zeros(1,2);

// CONTROL LAW COMPUTATION
for t=nh:-1:1
  fp=om+ones(4,1)*fs;         // penalty + reminder from last step
  // expectation
  f=sum((fp.*th),'c');          // expectation over y
  // minimization
  if f(1)<f(3),               // for y(t-1)=1
    us(t,1)=1; fs(1)=f(1);    // optimal control, minimum of criterion
  else
    us(t,1)=2; fs(1)=f(3);    // optimal control, minimum of criterion
  end
  if f(2)<f(4),               // for y(t-1)=2
    us(t,2)=1; fs(2)=f(2);    // optimal control, minimum of criterion
  else
    us(t,2)=2; fs(2)=f(4);    // optimal control, minimum of criterion
  end
end
J=fs(y0);                     // final value of criterion

// CONTROL APPLICATION
y(1)=y0;
for t=1:nh
  u(t+1)=us(t,y(t));          // optimal control
  y(t+1)=dSamp(u(t+1),y(t),th); // simulation
end
```

```
// RESULTS
plot(1:nh+1,y,'ro',1:nh+1,u,'g+')
set(gcf(),"position",[700 100 600 500])
legend('output','input');
set(gca(),'data_bounds',[.8 nh+.2, .8 2.2])
title('Optimal control with discrete model')

printf('\n Minimal value of the expectation of criterion: %g\n',J)
```

2.16 Control with state-space model

```
// T53ctrlX.sce
// Control with scalar 1st order regression model
// - simulated data  y(t)=a*y(t-1)+b*u(t)+k+e(t);
// - state realization of the model for synthesis
// - control on a single control interval with the length nd
// - following a settpoint s(t)
// Experiments
// - change penalizations of input(om) and input increments (la)
// - set new setpoint s
// ------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=100;                            // length of control interval
a1=.6; a2=-.2; b0=1; b1=.4; b2=-.1; k=-3; sd=.1; // regression model parameters
om=0; la=.1;                      // penalization (input, increment)
s=sign(10*sin(18*(1:nd)/nd));     // setpoint generation
// conversion to state-space model
M=[a1 b1 a2 b2 k
   0  0  0  0  0
   1  0  0  0  0
   0  1  0  0  0
   0  0  0  0  1];           // state matrix with set-point
N=[b0 1  0  0  0]';
Om=diag([1 om+la 0 la 0]);                // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;

S=list();
R=list();
R(nd+1)=zeros(Om);                 // initial condition for dyn. progr.

// CONTROL
// computation of control-law
for t=nd:-1:2
  Om(1,$)=-s(t); Om($,1)=-s(t); Om($,$)=s(t)**2;
  T=R(t+1)+Om;
  A=N'*T*N;
  B=N'*T*M;
  C=M'*T*M;
  S(t)=inv(A)*B;
  R(t)=C-S(t)'*A*S(t);
end

// control-law realization
y(1)=5; y(2)=-1;
u(1)=0; u(2)=0;
for t=3:nd
  u(t)=-S(t)*[y(t-1) u(t-1) y(t-2) u(t-2) 1]';     // optimaal control
  y(t)=a1*y(t-1)+a2*y(t-2)+b0*u(t)+b1*u(t-1)+b2*u(t-2)+k+sd*rand(1,1,'n');   // simulation
end

// RESULTS
x=1:nd;
```

```
plot(x,y(x),'--',x,u(x),x,s(x),':')
legend('y','u','s');
```

2.17 Adaptive control with state-space model

```
// T54ctrlXEst.sce
// Control with scalar 1st order regression model
// - simulated data  y(t)=a*y(t-1)+b*u(t)+k+e(t);
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - following a settpoint s(t)
// Experiments
// - change setpoint and system parameters (slow - quick system)
// - penalization of input variable
// - initial condition for estimation (better or worse initial param.)
// - length of control interval nh
// ----------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd('functions')

nd=100;                            // number of data to be controlled
ni=20;                             // length of pre-etimation
nh=15;                             // length of control interval
a1=.6;  a2=.2;  b0=1;   b1=-.4;  b2=.1; k=-3;  // parameters for simulation
sd=.1;                             // stdev for simulation
om=.01; la=.001;                   // penalization of input / increments

// PRE-ESTIMATION
V=1e-8*eye(7,7);                   // initial information matrix
ui(1:2)=zeros(1,2); yi(1:2)=zeros(1,2);
for t=3:ni
  ui(t)=rand(1,1,'n');
  yi(t)=a1*yi(t-1)+a2*yi(t-2)+b0*ui(t)+b1*ui(t-1)+b2*ui(t-2)+k+.01*rand(1,1,'n');
  Ps=[yi(t) yi(t-1) yi(t-2) ui(t) ui(t-1) ui(t-2) 1]';
  V=V+Ps*Ps';
end
Vyp=V(2:$,1); Vp=V(2:$,2:$); thi=inv(Vp)*Vyp; // point estimates
a1E=thi(1); a2E=thi(2); b0E=thi(3); b1E=thi(4); b2E=thi(5); kE=thi(6);
thi=[a1E a2E b0E b1E b2E kE];

s=sign(100*sin(18*(1:nd+nh)/(nd+nh)));  // set-point
y(1)=1; y(2)=-1;                   // initial output
u(1)=0; u(2)=0;                    // initial control

Om=diag([1 om+la 0 la 0]);         // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;
M=[a1E b1E a2E b2E kE
   0   0   0   0   0
   1   0   0   0   0
   0   1   0   0   0
   0   0   0   0   1];             // state-space model
N=[b0E 1   0   0   0]';            // state-space model

// COMPUTATION OF CONTROL-LAW
for t=3:nd          // loop for control
  R=0;                             // initial condition for dyn.prog.
  for i=nh:-1:1     // loop for receding horizon
    Om(1,$)=-s(t+i-1);
```

```
    Om($,1)=-s(t+i-1);
    Om($,$)=s(t+i-1)**2;
    T=R+Om;                        // dynamic
    A=N'*T*N;                      //    programming
    B=N'*T*M;
    C=M'*T*M;
    S=inv(A)*B;
    R=C-S'*A*S;
  end

// CONTROL REALIZATION (simulation)
  u(t)=-S*[y(t-1) u(t-1) y(t-2) u(t-2) 1]';             // optimal control value
  y(t)=a1*y(t-1)+a2*y(t-2)+b0*u(t)+b1*u(t-1)+b2*u(t-2)+k+sd*rand(1,1,'n');   // simulation

// ESTIMATION
  Ps=[y(t) y(t-1) y(t-2) u(t) u(t-1) u(t-2) 1]';
  V=V+Ps*Ps';
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp)*Vyp;                // point estimates
  a1E=th(1);                     // regression
  a2E=th(2);                     //    coefficients
  b0E=th(3);
  b1E=th(4);
  b2E=th(5);
  kE=th(6);
end            // of loop for control

// RESULTS
th=[a1 a2 b0 b1 b2 k];             // simulated parameters
thE=[a1E a2E b0E b1E b2E kE];      // estimated parameters
z=1:nd;
set(scf(),'position',[900 50 600 500])
plot(z,y(z),'--',z,u(z),z,s(z),':')
legend('y','u','s');

disp(th,'simulated parametrs')
disp(thi,'initial parametrs')
disp(thE,'estimated parametrs')
```

3 Supporting subroutines

3.1 Simulation of discrete data

3.2 Kalman filter

3.3 Coding of discrete variables