# Contents

# 1 Programs

## 1.1 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)

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

yt(1)=1; yt(2)=3;               // initial conditions for output
ut=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=[yt(t-1) yt(t-2) ut(t) ut(t-1) ut(t-2) 1]';
  // regression model
  yt(t)=th'*ps+s*rand(1,1,'norm');
end

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

## 1.2 Simulation with discrete model

```
// T13simDisc.sce
// SIMULATION OF DISCRETE MODEL
//  (multinomial model - controlled coin with memmory)
//         f( yt(t) | ut(t),yt(t-1) ),  yt,ut=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)

// PARAMETERS OF THE SIMULATION
// model parameter (conditional probabilities)
//yt(t)=1 =2  //  ut(t) yt(t-1)
// ---------------------------
th= [.2 .8    //  1    1
     .6 .4    //  1    2
     .9 .1    //  2    1
     .3 .7];  //  2    2
nd=50;                          // number of steps
ut=(rand(1,nd,'u')>.3)+1;       // control variable P(ut=1)=.3, P(ut=2)=.7
yt(1)=1;                        // initial condition for output

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

// RESULTS OF THE SIMULATION
subplot(211),plot(1:nd,ut,'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,yt,'b:.')
title('Output')
set(gca(),'data_bounds',[0 nd+1 .9 2.1])
```

### 1.3 Simulation with state-space model

```
// T15simState.sce
// SIMULATION WITH RM IN A STATE-SPACE FORM
// Experiments
// - extend to the 3-rd 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)
rand('seed',0)

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

et=rand(1,nd,'n');
ut=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*ut(t)+b*[ut(t-1) ut(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;

yt(1)=0; yt(2)=1;
xt(:,2)=[yt(2) ut(2) yt(1) ut(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*ut(t)+es;
  yt(t)=A*xt(:,t);
end

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

## 1.4 Least squares estimation

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

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

yt(1)=1; yt(2)=3;               // initial conditions for output
ut=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=[yt(t-1) yt(t-2) ut(t) ut(t-1) ut(t-2) 1]';
  // regression model
  yt(t)=th'*ps+s*rand(1,1,'norm');
end

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

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

scf(1);                         // comparison od output and prediction
plot(1:nd,yt,1:nd,yp)
legend('optput','prediction');
title('Verification of the estimates')
```

## 1.5 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)

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

yt(1)=1; yt(2)=3;               // initial conditions for output
ut=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=[yt(t-1) yt(t-2) ut(t) ut(t-1) ut(t-2) 1]';
  // regression model
  yt(t)=th'*ps+s*rand(1,1,'norm');
end

// ESTIMATION
V=1e-8*eye(7,7);                // initial information matrix
for t=3:nd
  psi=[yt(t:-1:t-2); ut(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
  thE(:,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

// Results
set(scf(1),'position',[500 60 600 500])
for i=1:6
  subplot(6,1,i),plot(thE(i,:))
  if i==1, title('Regression coefficients'), end
end
set(scf(2),'position',[50 60 400 200])
plot(r)
title('Noise variance')
```

**Estimation under model structure mismatch**

```
// T22estCont_B2.sce
// ESTIMATION OF 2ND ORDER REGRESSION MODEL
// The model for simulation differs from that for estimation
// i.e. MODEL STRUCTURE MISMATCH is tackled.
// - Bayesian on-line estimation with statistic update
// Experiments
// - try various input signals - set inp = 1,2,3,4
// - set different noise variance r=0,.1,1,10
// - define other regression coefficients a,b,k
// - you can also change the orders of models for simulation
//   as well as for estimation (adjust the beginning of loop)
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1)

// SIMULATION
// parameters
nd=100;                         // length of data
a=[.1 -.8 .3];                  // parameters at yt
b=[1 .2 -.5 -.8];               // parameters at ut
k=0;                            // constant (model absolute term)
s=.01;                          // noise variance
inp=1;                          // selection of input

yt(1)=0; yt(2)=0; yt(3)=0;      // initial conditions for output
                                            // inputs at disposal
select inp
case 1, ut=ones(nd,1);                      // one jump
case 2, ut=sin(10*%pi*(1:nd)/nd)';          // several jumps
case 3, ut=sign(10*sin(10*%pi*(1:nd)/nd))'; // sine wave
case 4, ut=.1*rand(nd,1,'n');               // white noise
end

// time loop
th=[a b k]';                    // vector of parameters
for t=4:nd
  // regression vector
  ps=[yt(t-1) yt(t-2) yt(t-3) ut(t) ut(t-1) ut(t-2) ut(t-3) 1]';
  // regression model
  yt(t)=th'*ps+s*rand(1,1,'norm');
end

// ESTIMATION
V=1e-8*eye(5,5);                // initial information matrix
for t=3:nd
  psi=[yt(t:-1:t-1); ut(t:-1:t-1); 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
```

```
  thE(:,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

// Results
set(scf(1),'position',[500 60 600 500])
for i=1:4
  subplot(6,1,i),plot(thE(i,:))
  if i==1, title('Regression coefficients'), end
end
set(scf(2),'position',[50 60 400 200])
plot(r)
title('Noise variance')
set(scf(3),'position',[50 360 400 200])
plot(1:nd,yt,1:nd,ut)
legend('output','input');
title 'Dataset for estimation'

disp(th','Simulated parameters')
disp(thE(:,$)','Easimated parameters')
```

• *Simulation of $1^{st}$ order model, estimation with the order that can be set*

```
// T22estCont_B3.sce
// ESTIMATION OF 2ND ORDER REGRESSION MODEL
// The model for simulation differs from that for estimation
// i.e. MODEL STRUCTURE MISMATCH is tackled.
// - Bayesian on-line estimation with statistic update
// Experiments
// - try various input signals - set inp = 1,2,3,4
// - set different noise variance r=0,.1,1,10
// - define other regression coefficients a,b,k
// - you can also change the orders of models for simulation
//   as well as for estimation (adjust the beginning of loop)
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1)

// SIMULATION
// parameters
nd=100;                         // length of data
a=[.1 ];                        // parameters at yt
b=[1 .2];                       // parameters at ut
k=0;                            // constant (model absolute term)
s=.01;                          // noise variance
inp=1;                          // selection of input
ord=3;                          // order of the estimated model

yt(1)=0; yt(2)=0; yt(3)=0;      // initial conditions for output
                                            // inputs at disposal
select inp
case 1, ut=ones(nd,1);                      // one jump
case 2, ut=sin(10*%pi*(1:nd)/nd)';          // several jumps
case 3, ut=sign(10*sin(10*%pi*(1:nd)/nd))'; // sine wave
case 4, ut=.1*rand(nd,1,'n');               // white noise
end

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

// ESTIMATION
nth=2*(ord+1)+1;
V=1e-8*eye(nth,nth);              // initial information matrix
for t=(ord+1):nd
  psi=[yt(t:-1:t-ord); ut(t:-1:t-ord); 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
  thE(:,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

// Results
set(scf(1),'position',[500 60 600 500])
for i=1:nth-1
  subplot(nth,1,i),plot(thE(i,:))
  if i==1, title('Regression coefficients'), end
end
set(scf(2),'position',[50 60 400 200])
plot(r)
title('Noise variance')
set(scf(3),'position',[50 360 400 200])
plot(1:nd,yt,1:nd,ut)
legend('output','input');
title 'Dataset for estimation'

disp(th','Simulated parameters')
disp(thE(:,$)','Easimated parameters')
```

• *Estimation of real data*

```
// T22estCont_B4.sce
// ESTIMATION OF 2ND ORDER REGRESSION MODEL
// Estimation with REAL DATA (intensities of traffic in Strahov tunnel)
// i.e. MODEL STRUCTURE MISMATCH is tackled.
// - Bayesian on-line estimation with statistic update
// - cannot be done without estimation !!!
// Experiments
// - change the order of model for estimation
// Result: estimated parameters (better with prediction - T32preCont_Adapt3.sce)
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1)

ord=5;                        // order of the estimated model
// selection of dataset
k=3;                          // selected day
nz=50;                        // beginning of the day
nd=288;                       // length of the whole one day data

// DATA
dtAll=csvRead('STRAHOV.csv',';');
dt=dtAll((k-1)*288+(nz+1:nd),1);
yt=dt(:,1);

// ESTIMATION
nth=ord+2;
V=1e-8*eye(nth,nth);              // initial information matrix
for t=(ord+1):(nd-nz)
  psi=[yt(t:-1:t-ord); 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
  thE(:,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

// Results
set(scf(1),'position',[500 60 600 500])
for i=1:nth-1
  subplot(nth,1,i),plot(thE(i,:))
  if i==1, title('Regression coefficients'), end
end
set(scf(2),'position',[50 60 400 200])
plot(r)
title('Noise variance')
set(scf(3),'position',[50 360 400 200])
plot(1:nd-nz,yt)
legend('output');
title 'Dataset for estimation'
```

```
disp(thE(:,$)','Easimated parameters')
```

## 1.6 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)

// SIMULATION
nd=500;                         // number of steps
// parameters of simulation
thS= [.2 .8
      .6 .4
      .9 .1
      .3 .7];
ut=(rand(1,nd,'u')>.3)+1;     // control variable P(ut=1)=.3, P(ut=2)=.7
yt(1)=1;                        // initial condition for output

// time loop of simulation
for t=2:nd
  i=2*(ut(t)-1)+yt(t-1);       // row in the model parameter
  yt(t)=(rand(1,1,'u')>thS(i,1))+1;  // generation of the output
end

// ESTIMATION
V=zeros(4,2);                   // initial statistics
for t=2:nd
  i=2*(ut(t)-1)+yt(t-1);      // row of model matrix
  V(i,yt(t))=V(i,yt(t))+1;    // updt of statistics
  thp=V./(sum(V,2)*ones(1,2)); // pt estimates of parameters
  thE(:,t)=thp(:,1);
end

// Results
set(scf(1),'position',[600 60 600 500])
for i=1:4
  subplot(4,1,i)
  plot(thE(i,:))                                  // estimated
  plot((nd-199:nd),ones(1,200)*thS(i,1),':r','linewidth',2)     // 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
```

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

## 1.7 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("ScIntro.sce",-1),mode(0)

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,'norm'); // input
y(1)=1; y(2)=-1;                     // prior data


// 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;                         // zero prediction for time = t (np=0)
  for j=1:np                         // loop of predictions for t+1,...,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,'norm');      // output generation
end

// Results
s=(np+3):(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')

RPE=variance(y(s)-yp(s))/variance(y)    // relative prediction error
```

## 1.8 Adaptive on-line prediction with continuous model

```
// T32preCont_Adapt.sce
// N-STEP PREDICTION WITH CONTINUOUS MODEL (WITH ESTIMATION)
// Experiments
// Change: - np = number of steps of prediction
//         - r  = noise variance (effect on estimation)
//         - th = model parametrs
//         - u  = input signal (effect on estimation)
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=100;                             // number of data
np=5;                               // length of prediction (np>=1)
nz=3;                               // starting time (ord+1)
//  b0 a1 b1  a2 b2 k
th=[1 .4 -.3 -.5 .1 1]';            // regression coefficients
r=.2;                               // noise variance
u=sin(4*%pi*(1:nd)/nd)+rand(1,nd,'norm'); // input
y(1)=1; y(2)=-1;                    // prior data
Eth=rand(6,1,'n');                  // prior parametrs

nu=zeros(4,2);
for t=nz:(nd-np)                    // time loop (on-line tasks)
  // prediction
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]'; // regression vector
  yy=ps'*Eth;                             // 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'*Eth;                           // 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,'norm');      // output generation

  // estimation
  Ps=[y(t) u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]';  // reg.vect. for estim.
  if t==nz, V=1e-8*eye(length(Ps)); end // initial information matrix
  V=V+Ps*Ps';                           // update of statistics
  Vp=V(2:$,2:$);
  Vyp=V(2:$,1);
  Eth=inv(Vp+1e-8*eye(Vp))*Vyp;         // point estimates
  Et(:,t)=Eth(:,1);                     // stor est. parameters
end

// Results
```

```
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)

set(scf(1),'position',[100 100 1200 400]);
subplot(121),plot(Et')
set(gca(),"data_bounds",[0 nd+1 -1 2])
title('Evolution of estimated parameters')
subplot(122)
s=(np+3):(nd-np);
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'])
```

**Prediction under model structure mismatch**

• *Prediction with $3^{rd}$ order model in simulation and $2^{nd}$ order one used for estimation*

```
// T32preCont_Adapt2.sce
// N-STEP PREDICTION WITH CONTINUOUS MODEL (WITh ESTIMATION)
// The model for simulation differs from that for estimation
// i.e. MODEL STRUCTURE MISMATCH is tackled.
// Experiments
// Change: - np = number of steps of prediction
//         - r  = noise variance (effect on estimation)
//         - th = model parametrs
//         - u  = input signal (effect on estimation)
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=100;                              // number of data
np=3;                                // length of prediction (np>=1)
//  b0 a1 b1  a2 b2 a3 b3 k
th=[1 .4 -.3 -.5 .1 .6 .1 1]';       // regression coefficients
r=.2;                                // noise variance
u=sin(4*%pi*(1:nd)/nd)+rand(1,nd,'norm'); // input
y(1)=1; y(2)=-1; y(3)=0;             // prior data
Eth=rand(8,1,'n');                   // prior parametrs

nu=zeros(4,2);
yp=ones(1,nd);
nz=4;
for t=nz:(nd-np)                     // time loop (on-line tasks)
  // prediction
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) y(t-3) u(t-3) 1]';  // regression vector
  yy=ps'*Eth;                        // 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'*Eth;                      // 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) y(t-3) u(t-3) 1]';  // regression vector for sim.
  y(t)=ps'*th+sqrt(r)*rand(1,1,'norm'); // output generation

  // estimation
  Ps=[y(t) u(t) y(t-1) u(t-1) y(t-2) u(t-2) y(t-3) u(t-3) 1]';  // reg.vect. for estim.
  if t==nz, V=1e-8*eye(length(Ps)); end // initial information matrix
  V=V+Ps*Ps';                          // update of statistics
  Vp=V(2:$,2:$);
  Vyp=V(2:$,1);
```

```
  Eth=inv(Vp+1e-8*eye(Vp))*Vyp;         // point estimates
  Et(:,t)=Eth(:,1);                     // stor est. parameters
end

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

set(scf(1),'position',[100 100 1200 400]);
subplot(121),plot(Et')
set(gca(),"data_bounds",[0 nd+1 -1 2])
title('Evolution of estimated parameters')
legend('b0E','a1E','b1E','a2E','b2E','kE');
subplot(122)
s=(np+nz):(nd-np);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[1 nd -6 10])
legend('output','prediction');
title([string(np),'- step ahead prediction'])
```

• *Prediction with real data and model of the optional order* ord *pre-set for 5*

```
// T32preCont_Adapt3.sce
// np-STEP PREDICTION WITH CONTINUOUS MODEL (WITh ESTIMATION)
// REAL DATA (intensity) from Strahov tunnel are used
// i.e. MODEL STRUCTURE MISMATCH is tackled.
// Experiments
// Change: - np  = number of steps of prediction
//         - ord = order of the model for estimation
// Result: - visual comparison of yt and yp
//         - RPE = relative prediction error
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

np=5;                               // length of prediction (np>=1)
ord=2;                              // order of the estimated model
// data selection
k=3;                                // which day is selected
nz=50;                              // beginning of the day
nd=288*2;                           // length of the whole one day data

// DATA
dtAll=csvRead('STRAHOV.csv',';');
dt=dtAll((k-1)*288+(nz+1:nd),1);
nth=ord+2;                          // size of V
V=1e-8*eye(nth,nth);                // initial information matrix
thE=rand(nth-1,1,'n');              // prior parametrs
yt=dt(1:ord);                       // prior data

for t=ord+1:(nd-np-nz)              // time loop (on-line tasks)
//  PREDICTION
  ps=[yt(t-1:-1:t-ord); 1];         // regression vector
  yy=ps'*thE;                       // 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=[yy; ps(1:(ord-1)); 1];      // reg. vector
    yy=ps'*thE;                     // new prediction (partial)
  end
  yp(t+np,1)=yy;                    // final prediction for time t+np

  // DATA MEASUREMENT (as if)
  yt(t)=dt(t);                      // measuring of output

  // ESTIMATION
  psi=[yt(t:-1:t-ord); 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
  thE(:,1)=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
  Et(:,t)=thE;                              // stor est. parameters
```

```
end

// Results
// evolution of parametrs
set(scf(1),'position',[100 100 1200 400]);
subplot(121),plot(Et')
set(gca(),"data_bounds",[0 nd+1 -1 5])
title('Evolution of estimated parameters')
legend('b0E','a1E','b1E','a2E','b2E','kE');
// comparison of yt and yp
subplot(122)
s=1:length(yt);
plot(s,yt(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[1 nd -4 max(yt)+5])
legend('output','prediction');
title([string(np),'- step ahead prediction'])

RPE=variance(yt(s)-yp(s))/variance(yt(s))
```

## 1.9 Prediction with discrete model

```
// T33preCat_Off.sce
// PREDICTION WITH DISCRETE MODEL (OFF-LINE), KNOWN 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)

nd=50;                              // length of data sample
np=0;                               // 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)=1;

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

// PREDICTION
yy=ones(1,nd);                      // fictitious predicted output
for t=2:(nd-np)
  i=2*(u(t)-1)+y(t-1);              // row of the table
  yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1;  // prediction generation
  for j=1:np
    i=2*(u(t+j)-1)+yy;              // row of the table
    yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1; // prediction
  end
  yp(t+np)=yy;                      // np-step predction
end

// Results
disp(th,' Model parameters'), disp(' ')

s=(np+3):nd;
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(:)~=yp(:)), From=nd
```

## 1.10 Adaptive prediction with discrete model

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

nd=50;                              // number of data
np=5;                               // length of prediction
th1=[0.8 0.1 0.4 0.7]';             // parameters for simulation (for y=1)
th=[th1 1-th1];                     // all parameters
u=(rand(1,nd)>.3)+1;                // input
y=ones(1,nd);

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

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

// PREDICTION
yy(1)=1;                            // fictitious predicted output
for t=2:(nd-np)
  i=2*(u(t)-1)+y(t-1);              // row of the table
  yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1; // prediction generation
  for j=1:np
    i=2*(u(t+j)-1)+yy;              // row of the table
    yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1; // prediction generation
  end
  yp(t+np)=yy;                      // np-step predction
end

// Results
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)
```

```
s=(np+3):nd;
plot(s,y(s),'.:',s,yp(s),'rx')
set(gcf(),'position',[300 100 500 400])
set(gca(),"data_bounds",[0 nd+1 .9 2.1])
legend('output','prediction');
title(string(np)+'-steps ahead prediction')

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

## 1.11 Adaptive on-line prediction with discrete model

```
// T35preCat_OnEst.sce
// PREDICTION WITH DISCRETE MODEL (ON-LINE)
// 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?
// Remark: another way og generation is
//            y(t)=sum(rand(1,1,'u')>cumsum(th(i,:)))+1;
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=150;                             // number of data
np=2;                               // length of prediction
th1=[0.98 0.01 0.04 0.97]';         // parameters (for y=1)
th=[th1 1-th1];                     // all parameters
u=(rand(1,nd+np,'u')>.3)+1;         // input
y(1)=1;

// TIME LOOP
nu=1e-8*ones(4,2);
Et=zeros(4,nd-np);
for t=2:nd                          // time loop
  // prediction
  i=2*(u(t)-1)+y(t-1);              // row of the table
  yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1; // prediction generation
  for j=1:np
    i=2*(u(t+j)-1)+yy;              // row of the table
    yy=sum(rand(1,1,'u')>cumsum(th(i,:)))+1; // prediction generation
  end
  yp(t+np)=yy;                      // np-step predction

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

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

// Results
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
```

```
disp(Eth)

s=np+2:np+51;
set(scf(),'position',[100 100 1000 400])
subplot(121),plot(Et')
title('Evolution of estimated parameters')
set(gca(),"data_bounds",[0 nd-np+1 -.1 1.1])
subplot(122),plot(s,y(s),s,yp(s),'.:')
title('First 50 outputs and their prediction')
set(gca(),"data_bounds",[s(1) s($) .9 2.1])

s=np+2:nd;
Wrong=sum(y(s)~=yp(s))
From=nd-np
```

## 1.12 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), getd()

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;
xt(:,1)=[0 0]';
ut=rand(1:nd,'n');
// time loop of simulation
for t=2:nd
  xt(:,t)=M*xt(:,t-1)+N*ut(t)+rw*rand(2,1,'n');
  yt(t)  =A*xt(:,t)+B*ut(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),yt(t),ut(t),M,N,[],A,B,[],Rw,Rv,Rx);
end

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

```
title('Output')
legend('output','estimate')
```

## 1.13 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 SCIHOME/ScIntro.sce, mode(0), getd()

// 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;
zt(:,1)=[0 0]';                           // initial state

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

// Results
plot(g(1,:),g(2,:),'k:')
plot(x(1,:),x(2,:),'b.')
plot(zt(1,:),zt(2,:),'r.:')
legend('signal','measurements','estimate');
```

## 1.14 Control with regression model

```
// T53ctrlX.sce
// Control with scalar 2nd order regression model
// - simulated data
//   y(t)=b0*u(t)+a1*y(t-1)+b1*u(t)+a2*y(t-2)+b2*u(t-2)+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)

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),':')
title 'Control with the 2nd order regression model'
legend('y','u','s');
```

## 1.15 Adaptive control with regression 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)

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')
```

## 1.16 Control with discrete model

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

// 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=[.9 .1   // 1 1
    .4 .6   // 1 2
    .8 .2   // 2 1
    .1 .9]; // 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)=dsim(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 Supporting subroutines

## 2.1 Simulation of discrete data

```
function y=dsim(u,y1,th)
  // y=dsim(u,y1,th)    Simulation of a disctrete system
  // y   new output
  // u   input
  // y1  old nput

  i=psi2row([u y1]); // index of conditional regressor
  yy=rand(1,1,'unif')<th(i,1);   // probability of conditional regressor
  y=2-yy;            // output (with values 1, 2)
endfunction
```

## 2.2 Kalman filter

```
function [xt,Rx,yp]=Kalman(xt,yt,ut,M,N,F,A,B,G,Rw,Rv,Rx)
  // Kalman filter for state estimtion with the model
  //                              xt = M*xt + N*ut + F + w
  //                              yt = A*xt + B*ut + G + v
  // xt    state
  // Rx    state estimate covariance matrix
  // yp    output prediction
  // yt    output
  // ut    input
  // M,N,F state model parameters
  // A,B,G output model parameters
  // Rw    state model covariance
  // Rv    output model covariance

  nx=size(M,1);
  ny=size(A,1);
  if isempty(F), F=zeros(nx,1); end
  if isempty(G), G=zeros(ny,1); end

  xt=M*xt+N*ut+F;                      // time update of the state
  Rx=Rw+M*Rx*M';                       // time updt. of state covariance

  yp=A*xt+B*ut+G;                      // output prediction
  Ry=Rv+A*Rx*A';                       // noise covariance update
  Rx=Rx-Rx*A'*inv(Ry)*A*Rx;            // state est. coariance update
  ey=yt-yp;                            // prediction error
  KG=Rx*A'*inv(Rv);                    // Kalman gain
  xt=xt+KG*ey;                         // data update of the state
endfunction
```

## 2.3 Coding of discrete variables

```
function i=psi2row(x,b)
  // i=psi2row(x,b)  i is row number of a model table with
  //                 the regression vector x with the base b;
  //                 elements of x(i) are 1,2,...,nb(i)
  // it is based on the relation
  //     i=b(n-1)b(n-2)...b(1)(x(n)-1)+...+b(1)(x(2)-1)+x(1)

  n=length(x);
  if argn(2)<2, b=2*ones(1,n); end
  bb=b(2:n);
  bb=bb(:)';
  b=[bb 1];
  i=0;
  for j=1:n
    i=(i+x(j)-1)*b(j);
  end
  i=i+1;
endfunction
```