Examples on Bayesian Statistics

Contents

1 Model of a driven car (very simplified) 2
- 1.1 Simulation . . . 2
- 1.2 Estimation with prediction . . . 3
- 1.3 Control . . . 4

2 New model of the car 8
- 2.1 Estimation, prediction . . . 10
- 2.2 Control . . . 12
- 2.3 Classification . . . 19

3 Mixture model of a driven car 21
- 3.1 Classification . . . 21
- 3.2 Prediction with mixture model . . . 23
- 3.3 Control . . . 25
- 3.4 Control with a mixture model . . . 30

4 Model of LOS (Level Of Service) 35
- 4.1 Simulation . . . 35
- 4.2 Estimation . . . 36
- 4.3 Prediction . . . 37
- 4.4 Control . . . 39
- 4.5 Simulation with a mixture model . . . 40
- 4.6 Estimation with a mixture model . . . 42
- 4.7 Prediction with the controlled intensity . . . 43

1 Model of a driven car (very simplified)

1.1 Simulation

We are considering a very simplified car model based on the dynamic law

$$\frac{ds}{dt} = v, \quad \frac{dv}{dt} = a$$

where s is the path (distance traveled), v is the speed and a is the acceleration.

After discretization (with the step h) we have

$$\frac{s_t - s_{t-1}}{h_v} = v_t, \quad \frac{v_t - v_{t-1}}{h_a} = a_t$$

The control variable is a_t. From the above relation we have the speed $v_t = v_{t-1} + h_a a_t$. The distance (if needed) is $s_t = s_{t-1} + h_v v_t$.

If $a_t > 0$, the car is accelerating, if $a_t < 0$, it is breaking, each action with the different maner (different efficiency - breaking is stronger than accelerating). The speed is limited to be $v_t \geq 0$ (the car does not go back) and $v_t \leq M$, where M is the highest allowed speed.

The air resistance is introduced like exponential forgetting. If the acceleration is zero, the speed is gradually diminishing with exponential decrease.

Here is the program of simulation

```
// Car1sim.sce
// Demonstration of the basic simulated car
//   s - path
//   v - speed
//   a - gas (acceleration)
// Experiments
// - change signal progression and check the output
//   (compare it with your expectations)
// - change the parameters inside the function "drive"
//   (make it more suitable to your expert knowledge)
// ---------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=200;
s=2; v=3; a=0;                // initial values
a=signal(nd,3);
for t=2:nd                    // simulation
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t));
end

// Results
s=1:nd;
```

```
set(scf(),'position',[300 200 900 400])
subplot(121),plot(s,v,s,a)
legend('speed','gas',2);
title('Speed and level of acceleration of a driven car')
subplot(122),plot(s,s,'r')
legend('path','s',2);
title('Path of the driven car')
```

and the procedure `drive`

```
function [s,v]=drive(s,v,a,k)
  // [s,v]=drive(s,v,a,k) - simulation of driven car
  //                        .. with air resistance
  // s    path
  // v    speed
  // a    acceleration (control variable)
  // k    parameters [step for accel., step for deccel.]
  if argn(2)<4, k=[.1 .3]; end
  if a>=0
    fi=500/(v+500), v=fi*v+k(1)*a; // accelerating
  else
    v=v+k(2)*a;                   // deccelerating (breaking)
  end
  if v<0, v=0; end                // limits
  if v>10, v=10; end              // .. on speed
  s=s+v;                          // path generation
endfunction
```

1.2 Estimation with prediction

If we want to predict speed (or distance) we need to model the car with regression model that provides the prediction. In the following program we first estimate the regression model describing the speed and then, with estimated parameters, we perform *np*-steps ahead prediction of the speed.

```
// Car2estpred.sce
// Estimation and prediction of the car speed
//   s - path
//   v - speed
//   a - gas (acceleration)
// model: v(t)=[v(t-1) a(t) 1]*th
// Experiments
// - change length for prediction
// ---------------------------------------
exec('ScIntro.sce',-1), mode(0)
```

```
nd=200;
np=0;
s=2; v=3; a=0;                // initial values
a=signal(nd,3);               // acceleration

for t=2:nd
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t));     // simulation
  y(t)=v(t);                  // collecting
  d(t,:)=[v(t-1) a(t) 1];     // .. data
end

disp('Estimated parameters')
th=inv(d'*d)*d'*y             // estimation

for t=2:nd-np
  vj=th(1)*v(t)+th(2)*a(t)+th(3);  // zero step prediction
  for j=1:np                       // loop for
  vj=th(1)*vj+th(2)*a(t+j)+th(3);  // .. more steps prediction
  end
  vp(t+np)=vj;                     // predicted value v(t+np)
end

set(scf(),'position',[500 100 600 400])
plot([v vp])
title('Speed and its prediction')
legend('speed','prediction',);
```

1.3 Control

PID control

First we are going to demonstrate PID control of the car speed or distance. In both cases the controlled variable has to follow the prescribed setpoint. The equation of the controller (for the continuous case) is

$$e(t) = y(t) - y_s(t)$$

$$u(t) = P \cdot e(t) + D \cdot \frac{de(t)}{dt} + I \cdot \int_0^t e(\tau)\, d\tau$$

where P, D, I are constants, $u(t)$ is control variable, $y(t)$ is controlled variable and $y_s(t)$ is setpoint. Derivative and integral are discretized in the program.

The variable to be controlled is selected via knob *ctrl*.

The program is here:

```
// Car3contPid.sce
// Control of the car speed/path with PID controller
//   p,i,d  // coefficients of PID regulator
//   s - path, v - speed, s - setpont
//   e - error between speed/path and its setpoint  stp
// Experiments
// - change the parametwrs p,i,d
// - change the setpoint stp
// - change the parametwrs ctrl (control od path/speed)
// -----------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=200;
ctrl=1;   // controlled: 1=speed, 2=distance
s=[0 0]; v=[0 0]; e=[0 0];    // initial conditions
if ctrl==1
  stp=(sign(sin(10*(1:nd)/nd))+1);
else
  stp=cumsum(sign(sin(10*(1:nd)/nd))+1);
end
p=5; i=0; d=0;               // setting of the controller

for t=3:nd          // xxxx TIME LOOP xxx
  if ctrl==1
    e(t-1)=v(t-1)-stp(t-1);
  else
    e(t-1)=s(t-1)-stp(t-1);
  end
  a(t)=-p*e(t-1)-i*sum(e)-d*(e(t-1)-e(t-2));
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t));
end

// Results
r=1:nd;
if ctrl==1
  plot(r,v,r,a)
  plot(r(1:5:$),stp(1:5:$),'mx','markersize',2)
  title('PID control of speed')
  legend('speed','control','setpoint',2);
else
  plot(r,s,r,a)
  plot(r(1:5:$),stp(1:5:$),'mx','markersize',2)
  title('PID control of distance')
  legend('distance','control','setpoint',2);
end
```

Optimal control

As a second variant of control, we consider the adaptive optimal control designed on a finite control interval using dynamic programming. Here, we proceed as follows:

1. With the data at disposal we pre-estimate the parameters of the model.
2. For the computed parameters we perform control design on an interval.
3. The control for the actual time instant (at the beginning of the interval), is applied to the system and the value of the output variable is generated.
4. With newly generated data we shift the interval and go to 1. This we repeat until the end time instant is reached.

The program controlling the speed with the 2nd order regression model is here:

```
// Car4contOpt.sce
// Control of car SPEED with scalar 2ND ORDER regression model
// - simulated data : from drive
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - speed following the setpoint st(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
// - try to make severe limits on v(t)
// ---------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=1000;            // number of data to be controlled
ni=150;              // length of pre-estimation
nh=50;              // length of control interval
om=.1; la=.1;      // penalization of input / increments
al=.99;

// PRE-ESTIMATION
V=1e-8*eye(7,7);                   // initial information matrix
ai(1:2)=zeros(1,2); yi(1:2)=zeros(1,2); vi=zeros(1,2);
for t=3:ni
  ai(t)=rand(1,1,'n');
  [yi(t),vi(t)]=drive(yi(t-1),vi(t-1),ai(t));

  Ps=[vi(t) vi(t-1) vi(t-2) ai(t) ai(t-1) ai(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];

st=signal(nd+nh,3,0,6,.01);       // setpoint for speed

s(1)=1; s(2)=-1;                   // initial output
a(1)=0; a(2)=0;                    // initial control
v(1)=0; v(2)=0;                    // init speed

Om=diag([1 om+la 0 la 0]);         // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;

// TIME LOOP FOR CONTROL
for t=3:nd
  // regression to state-space model
  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

  // OPTIMIZATION (backwards on control horizon)
  R=0;                             // initial condition for dyn.prog.
  for i=nh:-1:1     // loop for receding horizon
    Om(1,$)=-st(t+i-1);
    Om($,1)=-st(t+i-1);
    Om($,$)=st(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)
  a(t)=-S*[v(t-1) a(t-1) v(t-2) a(t-2) 1]';            // optimal control value
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t));

  // ESTIMATION
  Ps=[v(t) v(t-1) v(t-2) a(t) a(t-1) a(t-2) 1]';
  V=al*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
thE=[a1E a2E b0E b1E b2E kE];      // estimated parameters
r=1:nd;
set(scf(),'position',[900 50 600 500])
plot(r,v(r),'--',r,a(r),r,st(r),':')
legend('speed','control','setpoint',3);
//set(gca(),'data_bounds',[1,nd,-30,30])

disp('initial parametrs',thI)
disp('estimated parametrs',thE)
```

2 New model of the car

The variables included in the car model are:

- Speed v_t.
- Increment of the speed $d_t = v_t - v_{t-1}$
- Gear lever shift $k \in \underbrace{\{1,2,3,4\}}_{\text{acceleration}} \cup \underbrace{\{5,6,7\}}_{\text{idling}} \cup \underbrace{\{8,9,10\}}_{\text{breaking}}$
- Forgetting fi - air resistance. It increases with increasing speed.

The principle of the model is:

1. Calculate the forgetting which reflects the current speed.
2. Learn, if the speed growth is (i) positive (acceleration), (ii) almost zero (idling), (iii) negative (breaking). Based on this, we determine the gear ratio k that corresponds to the actual speed. It is given by the vectors α - driving force and m - the speed; e.g.

k	1	2	3	4	5	6	7	8	9	10
α	5	4	3	2	0	0	0	10	10	10
m	8	18	36	60	60	30	6	60	36	6
	acceleration				idling			breaking		

   Example

   difference $d > 0$, speed $v = 40$. Then we are in the section for acceleration and the closest speed is 36. So, the gear is 3 and we will use $al = 3$ for acceleration.

3. Finally we determine the new speed according to the relation: $v_t = v_{t-1} + \alpha \cdot g_t$ where g_t is the action variable - pressure on gas or brake pedal (zero

is for idling).

Remark

- *The action variable for driving is g which is determined by the driver (or here it is prescribed)*
- *The idling and breaking have three positions. Idling has zero force on engine while breaking has maximum force. The three positions are considered because these actions can take place at all possible speeds.*

  Remark: In fact, the speeds in m are expectations of the real speed. Then the real speed can be considered a mixture of random variables with these expectations and the variances such that the distributions a bit overlap. In this way the whole range of possible values of the speed is covered.
- *The search for the closest expected speed corresponding to the actual speed is based on the simple euclidean distance.*

The program of the newCar is here

```
function [vt,d,k]=newCar(v,d,g,al,m,vma,vmi,k1,k2,k3)
  // car with gear lever as the input
  // g    gas/break pedal (pressure)
  if argn(2)<9, k1=4; k2=7; k3=10; end

  // ADAPTIVE FORGETTING
  select 2     // 1-with, 2-without air resistance
  case 1
    ka=.095*vmi*vma/(vma-vmi);
    kv=.995-ka/vmi;
    fi=ka/(v+1)+kv;
  case 2
    fi=1;
  end
  im=length(m);

  // GEAR LEVER ESTIMATE
  if d>=0           // d>=0             acceleration
    k=amin(abs(m(1:k1)-v));
  else              // d<0
    if abs(g)<.3
      k=amin(abs(m((k1+1):k2)-v))+k1;   // idling
    else
      k=amin(abs(m((k2+1):im)-v))+k2;   // breaking
    end
  end

  // MOTION EQUATION
```

```
  vt=fi*v+al(k)*g;            // speed model
  vt=max(vt,0);               // non-negativity
  d=vt-v;                     // speed increment
endfunction
```

Now, we try to apply the basic tasks with this model. They are:
(*i*) estimation + prediction, (*ii*) control (PID + optimal), (*iii*) classification.

2.1 Estimation, prediction

The program is

```
// CarNew10est.sce
// Simulation, gear lever classification, estimation
// and prediction with the newCar model
// Estimated model is  Ps=[v(t+1),v(t),v(t-1),g(t),g(t-1),kE(t)].
//   However, Ps=[v(t+1),v(t),g(t),kE(t)]
//   and also Ps=[v(t+1),v(t),g(t)] is sufficient   ???
// Experiments
// - change the action variable g
// - check if al and m are chosen properly
// - change the common variance in the components
//     GaussN(v(t+1),m(k),1) - m(k) is expectation
// - check the estimated parameters
// ------------------------------------------------
exec('ScIntro.sce',-1); mode(2); format('v',6);
getd();

nd=200;
// gass pedal: (-1, 1) neg. is breaking, pos. acceleration
o=ones(1,20);
g=[10*o 1*o 8*o 0*o -5*o 8*o 0*o 5*o 3*o -5*o]/10;

//k=1  2  3  4     5  6  7     8   9  10
al=[5  4  3  2     0  0   0    10  10  5];   // force
m=[84  18 36 60    60 30  6    60  36  6];   // speed
//     gear         idle      break

v(2)=1; z(2)=.1; vmi=1; vma=80; im=length(m); d=0; kE=[1;1];
V=eye(6,6);

for t=2:nd

  // simulation
  [v(t+1),d,k(t)]=newCar(v(t),d,g(t),al,m,vmi,vma);
```

```
  // classification
  if d>0
    for j=1:4
      [mic,q(j)]=GaussN(v(t+1),m(j),1);
    end
    q=exp(q-max(q));
    kE(t)=amax(q);
  elseif abs(d)<1
    for j=1:3
      [mic,q(j)]=GaussN(v(t+1),m(j+4),1);
    end
    q=exp(q-max(q));
    kE(t)=amax(q)+4;
  else
    for j=1:3
      [mic,q(j)]=GaussN(v(t+1),m(j+7),1);
    end
    q=exp(q-max(q))+7;
    kE(t)=amax(q);
  end

  // estimation
  Ps=[v(t+1),v(t),v(t-1),g(t),g(t-1),kE(t)];
  V=V+Ps'*Ps;
  Vy=V(1,1);
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp)*Vyp;
  tht(:,t)=th;

  // prediction
  ps=Ps(2:$);
  vp(t+1)=ps*th;
end

s=1:nd;
set(scf(),'position',[300 50 800 600])
plot(s,v(s),'r',s,10*g(s)','k')
plot(s,kE(s),'m',s,m(kE(s)))
legend('v','g','kE','m(kE)');

scf();
plot(s,v(s),s,vp(s))
legend('v','vp');
```

The time loop has the following parts

1. Simulation of the newCar
2. Classification of the gear lever position (which is a part of estimation)
   (a) the proximity with the model $f\left(v_t|m_k\right)$ is evaluated for each k within the group of the gear lever position which is determined by d ($<$, 0,$>$ then zero).
   (b) the maximum proximity determines the point estimate of the gear lever position $\hat{k}$
3. Estimation of the car regression model

$$f\left(v_{t+1}|v_t, v_{t-1}, g_t, g_{t-1}, \hat{k}_t\right)$$

   where we use the point estimate $\hat{k}$ of the gear lever position k.
4. Prediction which is just simulation from the estimated model (with zero noise).

2.2 Control

Programs are here:

– Control with PID regulator

For a model $f\left(y_{t+1}|u_{t+1}, y_t \cdots\right)$ the PID regulator has the form

$$z_t = y_t - s_t$$

where s_t is the setpoint (the course which y should follow), and

$$u_{t+1} = Pz_t + D\left(z_{t.} - z_{t-1}\right) + I\sum_{i=1}^{t} z_i$$

where P, I and D are coefficients at proportional, derivative and integral closed-loops.

Remark

The PID regulator aims to maintain the output at zero. If the "output" is $y_t - s_t$ which is to be zero then $y_t = s_t$.

```
//CarNew21ctrlPid.sce
// Simulation and PID control with the newCar model
// Experiments
// - change the P, I, P coefficients (start with P)
// - verify the meaning of these coefficients
// -----------------------------------------------
```

```
exec('ScIntro.sce',-1); mode(2); format('v',6);

nd=200;
// gass pedal: (-1, 1) neg. is breaking, pos. acceleration
o=ones(1,20);
//g=[10*o 1*o 8*o 0*o -5*o 8*o 0*o 5*o 3*o -5*o]/10;

s=ones(1,nd);       // setpoint
s(70:nd)=5;
s(120:nd)=2;

//k=1  2  3  4    5  6  7    8   9   10
al=[5  4  3  2    0  0  0   10   10   5];   // force
m=[ 4  9 18 30   30 15  3   30  18  3]*2;   // speed
//     gear         idle      break

v(2)=1; z(2)=.1; vmi=1; vma=80;
im=length(m); sz=0; d(2)=0;

P=.06; D=.04; I=.03;      // PID

for t=2:nd

  // pid control
  z(t)=v(t)-s(t);        // z=y-s
  dz=z(t)-z(t-1);        // diff(z)
  sz=sz+z(t);            // int(z)
  g(t+1)=-1*(P*z(t)+D*dz+I*sz);   // PID

  // simulation
  [v(t+1),d,k(t)]=newCar(v(t),d,g(t+1),al,m,vmi,vma);

end

t=1:nd;
set(scf(),'position',[300 50 800 600])
plot(t,v(t),t,z(t),t,s(t),'--',t,10*g(t),'k')
legend('v','z','s','10g');
```

In the program, the time loop has only two parts

- Control which from past output constructs the control variable for the next step
- Simulation of the newCar with the input g - pressure on the gas/break pedal.

– Optimal control with a first order model

The control synthesis at each time step minimizes the criterion

$$J_t = (y_t - s_t)^2 + \omega u_t^2 + \lambda (u_t - u_{t-1})^2$$

so it is optimal in the sense of this criterion.

The synthesis (with regression model) is performed backwards from the end of the control interval against the time. At each step the formula for the optimal control is constructed and stored. After reaching the beginning of the control interval, the application of the control is performed from the beginning of the interval in the direction of time, using the stored recipes for the optimal control.

```
// CarNew22ctrl1.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()

nd=150;                              // number of data to be controlled
ni=15;                               // length of pre-etimation
nh=20;                               // length of control interval
om=.00005;                               // penalization of input / increments

//k=1  2  3  4    5  6  7    8   9   10
al=[5  4  3  2    0  0  0   10   10   5];   // force
m=[ 4  9 18 30   30 15  3   30  18  3]*2;   // speed
d=0; vmi=1; vma=100; test=0;


// PRE-ESTIMATION
V=1e-8*eye(5,5);                     // 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')+2;
  [yi(t),d,k(t)]=newCar(yi(t-1),d,ui(t),al,m,vmi,vma);
  Ps=[yi(t) yi(t-1) ui(t) ui(t-1) 1]';
  V=V+Ps*Ps';
end
Vyp=V(2:$,1); Vp=V(2:$,2:$); thI=inv(Vp)*Vyp; // point estimates
```

```
a1E=thI(1); b0E=thI(2); b1E=thI(3); kE=thI(4);
thI=[a1E b0E b1E kE];
test=1;

s=sign(100*sin(22*(1:nd+nh)/(nd+nh)))+1.2;  // set-point

y(1)=1; y(2)=-1;                    // initial output
u(1)=0; u(2)=0;                     // initial control

Om=diag([1 om 0]);          // matrix penalization

// TIME LOOP FOR CONTROL
for t=3:nd
  // regression to state-space model
  M=[a1E b1E kE
     0   0   0
     0   0   1];            // state-space model
  N=[b0E 1   0]';           // state-space model

  // OPTIMIZATION (backwards on control horizon)
  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) 1]';  // optimal control value
  y(t-1)=max(y(t-1),0);
  [y(t),d,k(t)]=newCar(y(t-1),d,u(t),al,m,vmi,vma);
                                            // control application
  // ESTIMATION
  Ps=[y(t) y(t-1) u(t) u(t-1) 1]';
  V=V+Ps*Ps';
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  th=inv(Vp)*Vyp;                   // point estimates
  a1E=th(1);                        // regression coefficients
  b0E=th(2);
```

```
  b1E=th(3);
  kE=th(4);
end             // of loop for control

// OFF-LINE PREDICTION
for t=3:nd
  yp(t)=a1E*y(t-1)+b0E*u(t)+b1E*u(t-1)+kE;
end


// RESULTS
thE=[a1E b0E b1E kE];       // estimated parameters
z=1:nd;
set(scf(1),'position',[600 350 400 300])
plot(z,y(z),'--',z,u(z),z,s(z),':')
legend('y','u','s');
title('Control')
set(scf(2),'position',[1000 350 400 300])
plot([y yp])
legend('y','yp');
title('Prediction')

disp('Initial parametrs',thI)
disp('Estimated parametrs',thE)
```

For the program to work properly it is necessary first of all to choose proper penalization coefficient ω (λ is not used here - it can be introduce only for model with the order higher then one - see next program CarNew23ctrl2.sce).

If the model parameters are unknown and must be taken from the estimated model (as in this case—we do not know the regression model that would accurately describe our car), a preliminary estimate must be made. Without it, the control can be unstable before the correct parameters are found.

Then a standard procedure for optimal control is realized. I.e.

1. The regression model is converted to the state-space form.
2. Backward construction of the control laws is performed.
3. Application of the optimal control runs in the direction of time
4. Having new data, we cam perform estimation of the regression model.
5. And finally the prediction of the output is realized (with the estimated parameters).

Notice

In this example it is clearly seen that we have two models. One represents reality - here it is the model of the car. Second model tries to describe the reality - here it is the regression model. The program works with the estimated regression model. The model of the car only presents the point of measurements.

– Optimal control with a second order model

This program is the same as the previous one but for regression model of the second order. Here, both the penalization coefficients ω and λ can be used.

Let us recall: ω (om) penalizes values of the control while λ (la) penalizes its increments. The use of λ eliminates control deviation in steady state, but causes the control process to be slower.

```
// CarNew23ctrl2.sce
// Control with scalar 2nd order regression model
// - simulated data
//   y(t)=a1*y(t-1)+a2y(t-2)+b0*u(t)+b1u(t-1)+b2u(t-2)+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 and its incerment (om,la)
// - initial condition for estimation (better or worse initial param.)
// - length of control interval nh
// ---------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=150;                              // number of data to be controlled
ni=15;                               // length of pre-etimation
nh=20;                               // length of control interval
om=.1; la=.8;                    // penalization of input / increments

//k=1  2  3  4    5  6  7    8   9   10
al=[5  4  3  2    0  0  0   10   10   5];   // force
m=[ 4  9 18 30   30 15  3   30  18  3]*2;   // speed
d=0; vmi=1; vma=100;


// 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')+2;
  [yi(t),d,k(t)]=newCar(yi(t-1),d,ui(t),al,m,vmi,vma);
  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(22*(1:nd+nh)/(nd+nh)))+1.2;  // 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;

// TIME LOOP FOR CONTROL
for t=3:nd
  // regression to state-space model
  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

  // OPTIMIZATION (backwards on control horizon)
  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) y(t-2) u(t-1) u(t-2) 1]';  // optimal control value
  y(t-1)=max(y(t-1),0);
  [y(t),d,k(t)]=newCar(y(t-1),d,u(t),al,m,vmi,vma);
                                                 // control application
  // 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 coefficients
  a2E=th(2);
  b0E=th(3);
  b1E=th(4);
  b2E=th(5);
  kE=th(6);
end            // of loop for control

// OFF-LINE PREDICTION
for t=3:nd
  yp(t)=a1E*y(t-1)+a2E*y(t-2)+b0E*u(t)+b1E*u(t-1)+b2E*u(t-2)+kE;
end


// RESULTS
thE=[a1E a2E b0E b1E b2E kE];      // estimated parameters
z=1:nd;
set(scf(1),'position',[600 350 400 300])
plot(z,y(z),'--',z,u(z),z,s(z),':')
legend('y','u','s');
title('Control')
set(scf(2),'position',[1000 350 400 300])
plot([y yp])
legend('y','yp');

disp('Initial parametrs',thI)
disp('Estimated parametrs',thE)
```

2.3 Classification

Here we are only concerned with the task of classification. Estimation with classification is presented in `Car5classMix.sce`.

We divided the shifting gears into 10 cases: acceleration (4 cases), idling (3 cases) and breaking (3 cases). Each case is associated with the model of the speed

$$f\left(v_t|m_k\right) = N_v\left(m_k, 1\right),\ k = 1, 2, \cdots, 10$$

Using these model we compute proximities and classify to the case with maximal proximity.

Remark

Normalization of the proximities to weights is mot necessary. Proximities have the same maximum as weights.

The task is solved in the following program

```
// CarNew30clas.sce
// Simulation and gear-classification of the newCar
```

```
// Experiments
// - change parameters g, al and m0
// - test the situation when m is corrupted with noise
// - change the variance at the components
// ------------------------------------------------
exec('ScIntro.sce',-1); mode(2); format('v',6);
//getd();

nd=80;
// gass pedal: (-1, 1) neg. is breaking, pos. acceleration
o=ones(1,10);
g=[10*o 1*o 8*o 0*o -5*o 8*o 0*o 5*o]/10;

//k=1 2  3  4    5  6  7    8   9   10
al=[5 4  3  2    0  0  0   10  10   5];   // force
m0=[8 18 36 60   60 30 6   60  36  6];    // speed
//     gear         idle      break

v=1; d=.1; vmi=1; vma=80; im=length(m0);

for t=1:nd
    m=m0+0*randn(1,im);       // noise added to speeds
  for i=1:im
    if m(i)<0, m(i)=1; end    // nonnegative speeds
  end

  // simulation
  [v(t+1),d,k(t)]=newCar(v(t),d,g(t),al,m,vmi,vma);

  // classification
  if d>0
    for j=1:4                 // for acceleration (d>0)
      [mic,q(j)]=GaussN(v(t+1),m(j),1);
    end
    q=exp(q-max(q));
    kE(t)=amax(q);
  elseif abs(d)<1             // for idling (d=0)
    for j=1:3
      [mic,q(j)]=GaussN(v(t+1),m(j+4),1);
    end
    q=exp(q-max(q));
    kE(t)=amax(q)+4;
  else                        // for breaking (d<0)
    for j=1:3
      [mic,q(j)]=GaussN(v(t+1),m(j+7),1);
    end
```

```
    q=exp(q-max(q))+7;
    kE(t)=amax(q);            // maximum likelihood
  end

end

s=1:nd;
set(scf(),'position',[300 50 1200 300])
plot(s,v(s),s,10*g(s),s,m(k),'.',s,k(s),'m',[1 nd],[0 0],'c--')
legend('v','10g','m(k)','k','0',3);

scf3(); plot(s,v(s),s,m(k(s)))
legend('v','m(k)');

scf2(); plot([k kE])
legend('k','kE');
```

3 Mixture model of a driven car

3.1 Classification

The model simulating the car works at least in two regimes: accelerating and breaking. The models differ in the dynamic - one slowly increasing the speed the second reduces the speed more quickly.

The task of classification decides, which data belong to which model. The classification is based on mixture model estimation with two components (one corresponds to acceleration and the second to breaking). The estimation runs as follows:

1. Substitute the actual data into both components with the actual parameter estimates and determine zero-step prediction (proximity).
2. After normalization to probabilities, we obtain the weights of the components.
3. Each component statistics is updated by the actual data with the corresponding weight.

The weights we obtained can be used for classification. We classify the actual data to the component which has greater weight.

The program is

```
// Car5classMix.sce
// Classification (class estimation) of driven car
// .. with mixture of regression models
// - component: v(t)=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th
// Experiments
//   change
```

```
//   - intial conditions for estimation
//   - proberties of the car (inside the function "drive")
// ------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=1200;
nc=2;
s=2; v=3; a=0;                // initial values
a=signal(nd,3,0,3,.01);
for t=2:nd
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t),[.1 1.3]);
end

V{1}=randu(7,7); V{2}=randu(7,7);
th{1}=v2th(V{1}); th{2}=v2th(V{2});

for t=3:nd
  for j=1:nc
    Ev=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th{j};
    [nic,qL(j)]=GaussN(v(t),Ev,.1);     // proximities
  end
  qE=exp(qL-max(qL));
  w=fnorm(qE);                          // weights of components
  wt(:,t)=w;
  Ps=[v(t) a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1];
  for j=1:nc
    V{j}=V{j}+w(j)*Ps'*Ps;
    th{j}=v2th(V{j});
  end
end
q=sign(a);
s=find(q==-1);
q(s)=2;

// Results
cp=amax(wt,1);
set(scf(),'position',[200 100 600 400]);
plot([cp' a v])
title('class, control and speed')
legend('class','control','speed');
set(scf(),'position',[800 100 600 400]);
plot(cumsum(v));
title('distance travelled')
```

3.2 Prediction with mixture model

Off-line simulation - estimation - prediction of driven car with a mixture model with two components is presented. The components should reflect the two modes of driving: accelerating and breaking (breaking has stronger dynamics).

The prediction can be computed for arbitrary many steps ahead (but only if it makes practical sense). Prediction is always done with the component which is more probable (see the pointer *pt*)

```
// Car6estpredMix.sce
// Estimation of driven car with mixture of reg.mod.
// - component: v(t)=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th
//   + prediction        ALL OFF-LINE
// Experiments
//   change
//   - the properties of the car (in "drive")
//   - the initial informattion matrix V

// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=200;
nc=2;
np=3;
al=.995;
s=2; v=3; a=0;                // initial values
a=signal(nd,3);

// SIMULATION
for t=2:nd
  [s(t),v(t)]=drive(s(t-1),v(t-1),a(t));
end

select 1       // choose initialization of V
case 1         // from previous successfull estimation
  Vi{1}=[
   383.98   41.42   386.18   45.06   387.62   50.93   134.75
   41.36    69.29   33.92    58.57   27.9     50.88  -2.63
   386.07   34.     389.91   39.55   392.63   47.11   136.65
   45.12    58.5    39.46    66.68   32.19    57.52  -3.82
   387.56   27.95   392.73   32.18   396.97   41.59   138.84
   51.04    50.98   47.02    57.41   41.59    67.22  -1.45
   134.75  -2.67    136.62  -3.72    138.83  -1.52    99.79  ];
  Vi{2}=[
   375.47   48.57   375.06   53.99   373.74   54.62   124.02
   48.56    40.24   45.06    35.76   41.97    29.78   15.9
```

```
   375.01   45.09   375.08   51.35   374.23   52.76   123.28
   53.92    35.73   51.35    42.98   47.76    36.82   17.45
   373.75   41.97   374.22   47.76   373.69   49.93   122.24
   54.67    29.88   52.81    36.88   49.9     42.52   15.26
   123.97   15.98   123.29   17.42   122.25   15.32   72.98  ];
case 2
  V{1}=randu(7,7); V{2}=randu(7,7);  // random initialization
end
for j=1:2
  V{j}=Vi{j}+.1*randu();
  th{j}=v2th(V{j});
end

// ESTIMATION
for t=3:nd
  for j=1:nc
    Ev=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th{j};
    [nic,qL(j)]=GaussN(v(t),Ev,.1);
  end
  qN=qL/max(qL);
  qE=exp(qN);
  w=fnorm(qE);
  if isnan(w), disp(t), break, end
  wt(:,t)=w;
  for j=1:nc
    Ps=[v(t) a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]';
    V{j}=al*V{j}+w(j)*Ps*Ps';
    th{j}=v2th(V{j});
  end
end

// PREDICTION
for t=3:nd-np
  // weights
  for j=1:nc
    Ev=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th{j};
    [nic,qL(j)]=GaussN(v(t),Ev,.1);   // proximities
  end
  pt=amax(qL);
  // prediction with the best compoent
  vj(1)=[a(t) v(t-1) a(t-1) v(t-2) a(t-2) 1]*th{pt};  // zero step prediction
  vj(2)=[a(t) vj(1) a(t-1) v(t-2) a(t-2) 1]*th{pt};   // one step prediction
  for j=2:np-1
    vj(j+1)=[a(t) vj(j) a(t-1) vj(j-1) a(t-2) 1]*th{pt}; // one step prediction
  end
  vp(t+np)=vj(np);                    // predicted value v(t+np)
```

```
end

// Results
r=3:nd;
cp=amax(wt,1);
set(scf(),'position',[200 10 600 400]);
plot([v(r) vp(r)])
title('Speed prediction')
legend('speed', 'prediction',3);

set(scf(),'position',[800 10 600 300]);
plot([s(r) cumsum(v(r))]);
title('Path prediction')
legend('parh', 'prediction',2);

set(scf(),'position',[800 400 600 300]);
plot(wt')
title('Evolution of weights')
```

3.3 Control

Here, the optimal control of the driven car is computed. The algorithm is based on the state-space model derived from the first order regression model

$$v_t = \theta_1 v_{t-1} + \theta_2 u_t + \theta_3 u_{t-1} + \theta_4 + e_t$$

and it follows the theory of dynamic programming.

```
// Car7cont1st.sce
// Control of car SPEED with scalar 1ST ORDER regression model
// - simulated data : from drive
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - speed following the settpoint r(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
// - try to make more severe limits on v(t)
// ---------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0),

nd=500;                              // number of data to be controlled
ni=15;                               // length of pre-etimation
nh=20;                               // length of control interval
```

```
om=.0; la=.05;                      // penal.: 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); vi=zeros(1,2);
for t=3:ni
  ui(t)=rand(1,1,'n');
  [yi(t),vi(t)]=drive(yi(t-1),vi(t-1),ui(t));

  Ps=[vi(t) vi(t-1) 0 ui(t) ui(t-1) 0 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];

r=abs(signal(nd+nh,3,0,6,.01));     // setpoint for speed

y(1)=1; y(2)=-1;                    // initial output
u(1)=0; u(2)=0;                     // initial control
v(1)=0; v(2)=0;     // init speed

Om=diag([1 om+la 0 la 0]);          // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;

// TIME LOOP FOR CONTROL
for t=3:nd
  // regression to state-space model
  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

  // OPTIMIZATION (backwards on control horizon)
  R=0;                              // initial condition for dyn.prog.
  for i=nh:-1:1     // loop for receding horizon
    Om(1,$)=-r(t+i-1);
    Om($,1)=-r(t+i-1);
    Om($,$)=r(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*[v(t-1) u(t-1) v(t-2) u(t-2) 1]';            // optimal control value
  [y(t),v(t)]=drive(y(t-1),v(t-1),u(t));

  // ESTIMATION
  if 1
  Ps=[v(t) v(t-1) 0 u(t) u(t-1) 0 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
end            // of loop for control

// RESULTS
thE=[a1E a2E b0E b1E b2E kE];      // estimated parameters
z=1:nd;
set(scf(),'position',[900 50 600 500])
plot(z,v(z),'--',z,u(z),z,r(z),':')
legend('path','control','setpoint');
set(gca(),'data_bounds',[1,nd,-4,4])

disp('initial parametrs',thI)
disp('estimated parametrs',thE)
```

Here is the same example but with the second order regression model

$$v_t = \theta_1 v_{t-1} + \theta_2 v_{t-2} + \theta_3 u_t + \theta_4 u_{t-1} + \theta_5 u_{t-2} + \theta_6 + e_t$$

```
// Car7cont2nd.sce
// Control of car SPEED with scalar 2ND ORDER regression model
// - simulated data : from drive
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - speed following the settpoint r(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
// - try to make severe limits on v(t)
// ---------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=500;                              // number of data to be controlled
ni=15;                               // length of pre-etimation
nh=20;                               // length of control interval
om=.0; la=.05;            // penalization of input / increments
al=.95;

// PRE-ESTIMATION
V=1e-8*eye(7,7);                     // initial information matrix
ui(1:2)=zeros(1,2); yi(1:2)=zeros(1,2); vi=zeros(1,2);
for t=3:ni
  ui(t)=rand(1,1,'n');
  [yi(t),vi(t)]=drive(yi(t-1),vi(t-1),ui(t));

  Ps=[vi(t) vi(t-1) vi(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];

r=abs(signal(nd+nh,3,0,6,.05));    // setpoint for speed

y(1)=1; y(2)=-1;                     // initial output
u(1)=0; u(2)=0;                      // initial control
v(1)=0; v(2)=0;     // init speed

Om=diag([1 om+la 0 la 0]);           // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;

// TIME LOOP FOR CONTROL
for t=3:nd
  // regression to state-space model
  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

  // OPTIMIZATION (backwards on control horizon)
  R=0;                               // initial condition for dyn.prog.
  for i=nh:-1:1     // loop for receding horizon
    Om(1,$)=-r(t+i-1);
    Om($,1)=-r(t+i-1);
    Om($,$)=r(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*[v(t-1) u(t-1) v(t-2) u(t-2) 1]';            // optimal control value
  [y(t),v(t)]=drive(y(t-1),v(t-1),u(t));

  // ESTIMATION
  if 1
  Ps=[v(t) v(t-1) v(t-2) u(t) u(t-1) u(t-2) 1]';
  V=al*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
end            // of loop for control

// Results
thE=[a1E a2E b0E b1E b2E kE];      // estimated parameters
z=1:nd;
set(scf(),'position',[900 50 600 500])
plot(z,v(z),'--',z,u(z),z,r(z),':')
legend('speed','control','setpoint',3);
set(gca(),'data_bounds',[1,nd,-5,5])

disp('initial parametrs',thI)
disp('estimated parametrs',thE)
```

3.4 Control with a mixture model

Here, the mixture model used for control synthesis is presented. In estimation, the weights of components are evaluated and the synthesis is performed for the component hat is more probable (using point estimate of the pointer).

First order components

```
// Car8contMix1st.sce
// Control of car SPEED with scalar 1ST ORDER regression model
// - simulated data : from drive
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - speed following the settpoint r(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
// - try to make more severe limits on v(t)
// ---------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0), getd()

nd=500;                              // number of data to be controlled
nc=2;                                // number of components
ni=15;                               // length of pre-etimation
nh=20;                               // length of control interval
om=.1;                               // penal.: inputs/increments
al=.95;

// PRE-ESTIMATION
for j=1:nc
  V{j}=1e-8*randu(5,5);              // initial V{j}
  th{j}=v2th(V{j});                 // initial th{j}
end
ui(1:2)=zeros(1,2); yi(1:2)=zeros(1,2); vi=zeros(1,2); // init. values
for t=2:ni
  ui(t)=rand(1,1,'n');                              // control
  [yi(t),vi(t)]=drive(ui(t),yi(t-1),vi(t-1));       //simulation
  ps=[ui(t) vi(t-1) ui(t-1) 1];                     // regression vector
  for j=1:nc
    [nic,qL(j)]=GaussN(vi(t),ps*th{j},.1);          // proximity
  end
  qN=qL-max(qL);                                    // pre-normalization
  w=fnorm(exp(qN));                                 // weights
  Ps=[vi(t) ui(t) vi(t-1) ui(t-1) 1]';              // ext.reg. vector
  for j=1:nc
```

```
    V{j}=V{j}+w(j)*Ps*Ps';                          // update of V
    th{j}=v2th(V{j});                              // point estimates
  end
end
for j=1:nc, thI{j}=th{j}; end                       // remember

r=abs(signal(nd+nh,3,0,6,.01));                     // setpoint for speed

y(1)=1; y(2)=-1;                                    // initial output
u(1)=0; u(2)=0;                                     // initial control
v(1)=0; v(2)=0;                                     // init speed

Om=diag([1 om 0]);                                  // matrix penalization

// TIME LOOP FOR CONTROL
for t=2:nd
  // classification
  ps=[u(t) v(t-1) u(t-1) 1];                        // regression vector
  for j=1:nc
    [nic,qL(j)]=GaussN(v(t),ps*th{j},.1);           // proximity
  end
  pt=amax(qL);                                      // best component

  // regression to state-space model
  M=[th{pt}(2:$)'
     0         0  0
     0         0  1];     // state model
  N=[th{pt}(1) 1  0]';    // output model

  // OPTIMIZATION (backwards on control horizon)
  R=0;                              // initial condition for dyn.prog.
  for i=nh:-1:1          // loop for receding horizon
    Om(1,$)=-r(t+i-1);              // setpoint
    Om($,1)=-r(t+i-1);              // .. to
    Om($,$)=r(t+i-1)**2;            // .. penalization
    T=R+Om;                         // dynamic
    A=N'*T*N;                       // .. programming
    B=N'*T*M;
    C=M'*T*M;
    S=inv(A)*B;                     // optimal control law
    R=C-S'*A*S;
  end

  // CONTROL REALIZATION (simulation)
  u(t+1)=-S*[v(t) u(t) 1]';                  // optimal control value
  [y(t+1),v(t+1)]=drive(y(t),v(t),u(t+1));   // simulation
```

```
  // ESTIMATION
  if 1        // set to 0 for fixed parameters
  ps=[u(t+1) v(t) u(t) 1];                    // regression vector
  for j=1:nc
    [nic,qL(j)]=GaussN(v(t+1),ps*th{j},.1);  // proximity
  end
  qN=qL-max(qL);                              // pre-normalization
  w=fnorm(exp(qN));                           // weights
  Ps=[v(t+1) u(t+1) v(t) u(t) 1]';            // ext.reg. vector
  for j=1:nc
    V{j}=al*V{j}+w(j)*Ps*Ps';                 // update of V
    th{j}=v2th(V{j});                        // point estimates
  end
  end
end            // of loop for control

// Results
z=1:nd;
set(scf(),'position',[900 50 600 500])
plot(z,v(z),'--',z,u(z),z,r(z),':')
legend('path','control','setpoint');
set(gca(),'data_bounds',[1,nd,-4,4])
```

Second order components

```
// Car8contMix2nd.sce
// Control of car SPEED with scalar 2ND ORDER regression model
// - simulated data : from drive
// - state realization of the model for synthesis
// - control on a receding control interval with the length nh
// - speed following the settpoint r(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
// - try to make more severe limits on v(t)
// ---------------------------------------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=500;                          // number of data to be controlled
nc=2;                            // number of components
ni=50;                           // length of pre-estimation
nh=20;                           // length of control interval
```

```
om=.2; la=.5;                        // penal.: inputs/increments
al=.95;                              // exponential forgetting (for parameters)

// PRE-ESTIMATION
for j=1:nc
  V{j}=1e-8*randu(7,7);              // initial V{j}
  th{j}=v2th(V{j});                 // initial th{j}
end
ui(1:2)=zeros(1,2); yi(1:2)=zeros(1,2); vi=zeros(1,2);  // init. values
for t=3:ni
  ui(t)=rand(1,1,'n');                              // control
  [yi(t),vi(t)]=drive(ui(t),yi(t-1),vi(t-1));       // simulation
  ps=[ui(t) vi(t-1) ui(t-1) vi(t-2) ui(t-2) 1];     // reg. vector
  for j=1:nc
    [nic,qL(j)]=GaussN(vi(t),ps*th{j},.1);          // proximity
  end
  qN=qL-max(qL);                                    // pre-normalization
  w=fnorm(exp(qN));                                 // weights
  wtI(:,t)=w;
  wt(:,t)=w;                                        // remember
  Ps=[vi(t) ui(t) vi(t-1) ui(t-1) vi(t-2) ui(t-2) 1]'; // ext.reg. vector
  for j=1:nc
    V{j}=V{j}+w(j)*Ps*Ps';                          // update of V
    th{j}=v2th(V{j});                              // point estimates
  end
end
for j=1:nc, thI{j}=th{j}; end                       // remember

r=abs(signal(nd+nh,3,0,6,.02));  // setpoint for speed

y(1)=1; y(2)=-1; y(3)=0;         // initial output
u(1)=0; u(2)=0;  u(3)=0;         // initial control
v(1)=0; v(2)=0;  v(3)=0;         // init speed

Om=diag([1 om+la 0 la 0]);       // matrix penalization
Om(2,4)=-la; Om(4,2)=-la;

// TIME LOOP FOR CONTROL
for t=3:nd
  // classification
  ps=[u(t) v(t-1) u(t-1) v(t-2) u(t-2) 1];        // regression vector
  for j=1:nc
    [nic,qL(j)]=GaussN(v(t),ps*th{j},.1);         // proximity
  end
  pt=amax(qL);                                    // best component
```

```
  // regression to state-space model
  M=[th{pt}(2:$)'
     0         0  0  0  0
     1         0  0  0  0
     0         1  0  0  0
     0         0  0  0  1];        // state model
  N=[th{pt}(1) 1  0  0  0]';       // output model

  // OPTIMIZATION (backwards on control horizon)
  R=0;                             // initial condition for dyn.prog.
  for i=nh:-1:1                    // loop for receding horizon
    Om(1,$)=-r(t+i-1);             // setpoint
    Om($,1)=-r(t+i-1);             // .. to
    Om($,$)=r(t+i-1)**2;           // .. penalization
    T=R+Om;                        // dynamic
    A=N'*T*N;                      // .. programming
    B=N'*T*M;
    C=M'*T*M;
    S=inv(A)*B;                    // optimal control law
    R=C-S'*A*S;
  end

  // CONTROL REALIZATION (simulation)
  u(t+1)=-S*[v(t) u(t) v(t-1) u(t-1) 1]';    // optimal control value
  [y(t+1),v(t+1)]=drive(y(t),v(t),u(t+1));   // simulation

  // ESTIMATION
  if 1         // set to 0 for fixed parameters
  ps=[u(t+1) v(t) u(t) v(t-1) u(t-1) 1];
  for j=1:nc
    [nic,qL(j)]=GaussN(v(t+1),ps*th{j},.1);  // proximity
  end
  qN=qL-max(qL);                             // pre-normalization
  w=fnorm(exp(qN));                          // weights
  Ps=[v(t+1) u(t+1) v(t) u(t) v(t-1) u(t-1) 1]';  // ext.reg. vector
  for j=1:nc
    V{j}=al*V{j}+w(j)*Ps*Ps';                // update of V
    th{j}=v2th(V{j});                       // point estimates
  end
  end
end            // of loop for control

// Results
z=1:nd;
set(scf(),'position',[900 50 600 500])
plot(z,v(z),'--',z,u(z),z,r(z),':')
```

```
legend('path','control','setpoint');
set(gca(),'data_bounds',[1,nd,-4,4])
```

4 Model of LOS (Level Of Service)

4.1 Simulation

We want to simulate the level of service (traffic load indexed by 1 (low traffic), 2 (medium traffic), 3 (heavy traffic) in a traffic mini-region. We assume that it depends on its previous value and also on the car intensity coming to the region. Further we assume that this intensity ca be controlled by us so its values are 1 (low intensity), 2 (high intensity). Relations between variables are given by the categorical model $f\left(L_t|I_t, L_{t-1}\right)$, where L denotes LOS and I intensity. The model is defined by the table

I_t	1	1	1	2	2	2
L_{t-1}	1	2	3	1	2	3
$L_t = 1$	0.8	0.3	0.3	0.2	0.2	0.2
$L_t = 2$	0.1	0.6	0.3	0.4	0.3	0.2
$L_t = 3$	0.1	0.1	0.4	0.4	0.5	0.6

Program

```
// Los1sim.sce
// Simulation of LOS
// Experiments
//   change
//   - the model - conditional probability f(L(t)|I(t),L(t-1))
//   - the amount of errors (by the command "if randu()>.3, Lp=Lp+r; end")
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=1000;
I=ceil(.99*sin(6*%pi*(1:nd)/nd)+.001)+1;
//I=ones(1,nd)+1;
//  1  1  1  2  2  2    .. I(t)   intensity  (1,2,3)
//  1  2  3  1  2  3    .. L(t-1) last level
f=[.8 .3 .3 .2 .2 .2   // L(t)=1  level low
   .1 .6 .3 .4 .3 .2   // L(t)=2  level medium
   .1 .1 .4 .4 .5 .6]; // L(t)=3  lebel high
// dependence of the level on the intensities I1 and I2
L(1)=1;   // initial level

for t=2:nd
  z=xt2col([I(t),L(t-1)],[2,3]);        // coded vars
  Lp=amax(f(:,z));                      // max prob. for L
  if randu()>.5, r=1; else r=-1; end    //
```

```
  if randu()>.3, Lp=Lp+r; end           // error
  if Lp<1, Lp=1; end                    //
  if Lp>3, Lp=3; end                    //
  L(t)=Lp;                              // final L(t)
end

// Results
plot(L,'b.')
plot([1 I],'rx')
title(' b-LOS, r-intensity')

save simCat.dat I L f
```

4.2 Estimation

With the simulated data perform estimation of the model of LOS.
Program

```
// Los2est.sce
// Estimation of LOS
// - based on the simulation simCat.dat I L f
// Experiments
// - try to change the initial matrix V (e.g. from the predious
//     succesfull experimet)
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

load simCat.dat I L f;              // data from  Los1sim.sce
nd=length(L);

V=zeros(3,6);                       // initial statistics
for t=2:nd
  z=xt2col([I(t),L(t-1)],[2,3]);
  V(L(t),z)=V(L(t),z)+1;            // update of the statistics
end
th=fnorm(V,1);

// Results
disp('simulated')
sim_par=f
disp('estimated')
est_par=rn(th)

save estCat.dat th;
```

4.3 Prediction

Prediction of a discrete variable values can be interpreted as classification of independent variables into classes indexed by the estimated value of the target variable. Here, the couples $[I_t, L_{t-1}]$ are classified into the classes determined by the value of L_t

The prediction can be performed with the known (or pre-estimated parameters) off-line or on-line with the parameters continuously estimated. Both the cases are considered in the following programs:

Off-line prediction

```
// Los3predOff.sce
// Prediction of LOS (off-line)
// Experiments
// - change the length of prediction "np"
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

load simCat.dat  I L f;       // form  Los1sim.sce
load estCat.dat th;           // from  Los2est.sce

nd=length(L);
np=3;                         // length of prediction
// TIME LOOP
for t=2:(nd-np)
  // prediction
  i=2*(I(t)-1)+L(t-1);             // column of model matrix
  Lj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  for j=1:np                       // inner loop of prediction
    i=2*(I(t+j)-1)+Lj;             // column of model matrix
    Lj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // prediction
  end
  Lp(t+np)=Lj;                     // remember the last prediction
end

// Results
accuracy=sum(L==Lp)/nd

set(scf(),'position',[500 200 600 450]);
s=.9*nd:nd;
plot(s,L(s),'b.',s,Lp(s),'rx')
```

On-line prediction

```
// Los3predOn.sce
// Prediction of LOS (on-line) /init needed/
// Experiments
// - change somehow the initial V
// - thy to use the resulting V from  Los2est.sce
//     as the initial V in this program
// - change the length of prediction "np'
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

load simCat.dat  I L f;       // from  Los1sim.sce

nd=length(L);
np=3;
V=zeros(3,6);
V=randu(3,6);
th=fnorm(V,1);
// TIME LOOP
for t=2:(nd-np)
  // prediction
  i=2*(I(t)-1)+L(t-1);             // column of model matrix
  Lj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  for j=1:np                       // inner loop of prediction
    i=2*(I(t+j)-1)+Lj;             // column of model matrix
    Lj=sum(cumsum(th(:,i))<rand(1,1,'u'))+1; // prediction
  end
  Lp(t+np)=Lj;                     // remember the last prediction
  // measuring of data
  Lt=L(t);
  It=I(t);
  // estimation
  z=xt2col([I(t),L(t-1)],[2,3]);
  V(L(t),z)=V(L(t),z)+1;
  th=fnorm(V,1);
end


// Results
accuracy=sum(L==Lp)/nd

set(scf(),'position',[500 200 600 450]);
s=.9*nd:nd;
plot(s,L(s),'b.',s,Lp(s),'rx')
```

4.4 Control

The optimal control with the categorical model runs according to the dynamic programming, similarly as for the regression model, however, the algorithm is rather complex for programming. Here, we show a heuristic algorithm that toes not take care about the future. We are going to show it on example.

Model

y_{t-1}	1	1	2	2
u_t	1	2	1	2
$y_t = 1$	0.9	0.3	0.4	0.2
$y_t = 2$	0.1	0.7	0.6	0.8

Now, let the setpoint be

$$s_t = [1, 2, 1, 1, 2, \cdots]$$

ant the last $y_0 = 2$. Now, at time 1 after substituting $y_1 = s_1$ with $y_0 = 2$, the model reduces to

y_{t-1}	2	2
u_t	1	2
$y_1 = 1$	0.4	0.2

from which we select control with higher probability, that is $u_1 = 1$.

This control is applied and new value y_1 is generated.

Program

```
// Los4cont.sce
// Simulation and control of LOS (heuristic, one step ahaead)
// Experiments
//   change gen, stp, amount of errors in  "if randu()>.8, Lp=Lp+e; end"
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=300;
gen=1;                    // generation: 1=deterministic, 2=stochastic

//  1  1  2  2    .. I(t)
//  1  2  1  2    .. L(t-1)
f=[.8 .6 .3 .2    // L(t)    model: L(t)=f( I(t),L(t-1) )
   .2 .4 .7 .8];

stp=[2*ones(1,100) 1*ones(1:100) 2*ones(1,100)];    // setpoint
L=[1 1]; I=[1 1];

b=[1 2 1 2];
for t=2:nd                // xxx TIME LOOP xxx
  // CONTROL
  r=f(stp(t),:);          // row corresponding to stp
```

```
  c=r(b==L(t-1));        // columns corresponding to L(t-1)
  cm=amax(c);            // higher probability
  I(t)=b(cm);            // I(t) corresponding to higher probability

  // SIMULATION
  z=xt2col([I(t),L(t-1)],[2,2]);
  select gen
  case 1, Lp=amax(f(:,z));
  case 2, Lp=cumsx(f,z);
  end
  if randu()>.5, e=1; else e=-1; end    // error
  if randu()>1, Lp=Lp+e; end  // !randu()>1" means no errors
  if Lp<1, Lp=1; end
  if Lp>2, Lp=2; end
  L(t)=Lp;     // it is L(t-1) for the next step
end

// Results
plot(L,'bx')
plot(stp,'r.')
plot(I,'g-')
title(' b-LOS, r-setpoint, g-control')
```

4.5 Simulation with a mixture model

Here we consider more realistic situation when the traffic state changes. For simplicity, we assume that there are two different traffic states that can be described by two different models - components.

The program for simulation is here with the options

par selects parameters of the simulated models (1-standard, 2-less uncertainty, 3-deterministic)

err decides if a random error is added to the simulated LOS

genI selects frequency of changing the intensity I

```
// Los5simMix.sce
// Simulation a mixture of LOS
//   par = choice of the model
//   genI = type of model
// Experiment
// - try to implement stochastic simulation using commant  "cumsum"
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)
```

```
nd=1000;
par=1;
genI=2
select genI
case 1
I=ceil(.99*sin(6*%pi*(1:nd)/nd)+.001)+1;
case 2
I(1)=1;
for i=2:nd
  I(1,i)=I(1,i-1);
  if randu()<.4, I(i)=3-I(i); end
end
end
//     1  1  1  2  2  2    .. I
//     1  2  3  1  2  3    .. L
select par
case 1
f{1}=[.8 .4 .4 .3 .3 .3
      .1 .5 .3 .3 .3 .2
      .1 .1 .3 .4 .4 .5];
f{2}=[.7 .2 .2 .1 .1 .1
      .1 .6 .3 .4 .3 .2
      .2 .2 .5 .5 .6 .7];
case 2
f{1}=[.9 .8 .7 .3 .2 .1
      .1 .1 .2 .1 .1 .1
      .0 .1 .1 .6 .7 .8];
f{2}=[.4 .1 .1 .1 .1 .0
      .5 .7 .8 .7 .3 .3
      .1 .2 .1 .2 .6 .7];
case 3
f{1}=[ones(1,6); zeros(2,6)];
f{2}=[zeros(2,6); ones(1,6)];
end
L=[1 1]; c=[1 1];

// SIMULATION
for t=2:nd
  c(t)=(randu()>.4)+1;                  // pointer
  z=xt2col([I(t),L(t-1)],[2,3]);
  Lp=amax(f{c(t)}(:,z));                // deterministic simulation
  if randu()>.5, r=1; else r=-1; end
  if randu()>.3, Lp=Lp+r; end           // randu()>1  means no reeors
  if Lp<1, Lp=1; end
  if Lp>3, Lp=3; end
  L(t)=Lp;
```

```
end

// Results
s=1:100;
plot(L(s),'b.')
plot(I(s),'rx')
title(' b-LOS, r-intensity')
legend('L','I');

save simCatMix.dat I L f c par
```

4.6 Estimation with a mixture model

The choice of the model is
ini a way of initialization (1-random, 2-exact from simulation)

```
// Los6estMix.sce
// Estimation of a mixture of LOS (classification)
//   ini = selection of initialization
// Experiments
// -
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

load simCatMix.dat I L f c par;    // data from  Los1sim.sce
nd=length(L);
nc=max(c);                         // number of components
ka=1;

ini=2;
select ini
case 1, V{1}=randu(3,6);     V{2}=randu(3,6);     // random
case 2, V{1}=ka*f{1};           V{2}=ka*f{2};           // from simulation
end
th{1}=fnorm(V{1},1); th{2}=fnorm(V{2},1);
for t=2:nd
  z=xt2col([I(t),L(t-1)],[2,3]);
  for j=1:nc
    q(j)=th{j}(L(t),z);            // proximities
  end
  w=fnorm(q);                      // weights
  wt(:,t)=w;
  for j=1:nc
    V{j}(L(t),z)=V{j}(L(t),z)+w(j); // update of statistics
    th{j}=fnorm(V{j},1);           // point parameters
  end
```

```
end

// Results
cp=amax(wt,1);
q=c2c(c,cp);
accClass=acc(c,q(cp))
```

4.7 Prediction with the controlled intensity

Here, we assume, that the input intensity is on-line generated by a model

$$g\left(I_t|C_t\right)$$

where C_t is a given sequence, given e.g. by some authority. Then we use the Los model

$$f\left(L_t|I_t, L_{t-1}\right)$$

exactly as in the previous experiments.

Remark

The overall model is

$$h\left(I_t, L_t|C_t, L_{t-1}\right) = f\left(L_t|I_t, L_{t-1}\right) g\left(I_t|C_t\right)$$

And

$$h_m\left(L_t|C_t, L_{t-1}\right) = \sum_I f\left(L_t|I_t, L_{t-1}\right) g\left(I_t|C_t\right)$$

```
// Los8simestpred.sce
// Simulation of LOS with control  C -> I + estimation and prediction
//   All together
// Experiments   as in previous examples
// -------------------------------------
exec('ScIntro.sce',-1), mode(0)

nd=300;                              // number of data
np=0;                                // numb. of steps for prediction
nc=2;                                // nimb. of components
gen=0;                               // generation: 1=stochatic, 0=determnistic
genI=0                               // I generated: 1=on-line, 0=of-fline

// f(L(t)|I(t),L(t-1)
//  1  1  1  2  2  2    .. I
//  1  2  3  1  2  3    .. L
f=[.8 .3 .3 .2 .2 .6
   .1 .6 .3 .4 .3 .2
   .1 .1 .4 .4 .5 .2];   // set your own expert parameters !!
```

```
// f(I(t)|C(t)
//  1 2  .. C
g=[.85 .3
   .15 .7];              // model of I-control  (C -> I)

// setpoint
C=signalD(nd,2,.9);      // generation of control signal for I (C -> I)
if genI==0, for t=1:nd, I(t)=cumsx(g,C(t)); end, end
                         // off-line generation of I
L(1:2)=1; I(1:(2+np))=1; pt=1; Lp(1:(np+1))=1;
for j=1:nc
  V{j}=randu(3,6);
  th{j}=fnorm(V{j},1);
end

for t=2:(nd-np)     // xxxxxxxxxxxx  TIME LOOP  xxxxxxxxxxxxxxxxxxxxx
  if genI==1, I(t:(t+np))=cumsx(g,C(t));  end
                         // on-line generation of I

  // PREDICTION
  i=xt2col([I(t),L(t-1)],[2 3]);
  if gen==1
    Lj=sum(cumsum(th{pt}(:,i))<rand(1,1,'u'))+1; // zero-step prediction
  else
    Lj=amax(th{pt}(:,i));
  end
  for j=1:np                        // inner loop of prediction
    i=xt2col([I(t+j),Lj],[2 3]);
    if gen==1
      Lj=sum(cumsum(th{pt}(:,i))<rand(1,1,'u'))+1; // prediction
    else
      Lj=amax(th{pt}(:,i));
    end
  end
  Lp(t+np)=Lj;                      // remember the last prediction

  // SIMULATION
  z=xt2col([I(t),L(t-1)],[2 3]);
  Lt=amax(f(:,z));
  if randu()>.5, r=1; else r=-1; end
  if randu()>1, Lt=Lt+r; end            // disturbance
  if Lt<1, Lt=1; end                    // lower limit
  if Lt>3, Lt=3; end                    // upper limit
  L(t)=Lt;
```

```
  // ESTIMATION
  b=[2 3];
  z=xt2col([C(t) L(t-1)],b);
  for j=1:nc
    q(j)=th{j}(L(t),z);
  end
  w=fnorm(q);
  wt(:,t)=w;
  for j=1:nc
    V{j}(L(t),z)=V{j}(L(t),z)+w(j);
    th{j}=fnorm(V{j},1);
  end
  pt=amax(w);
end          // xxxxxxxxxxxx  END OF TIME LOOP  xxxxxxxxxxxxxxxxxxxxxx

// Results
s=(np+2):(nd-np);
accuracy=acc(L(s),Lp(s))

set(scf(),'position',[800 100 600 400]);
plot(L,'b.')
plot(Lp,'rx')
title('b-LOS, r-intensity')
legend('Los','prediction');

set(scf(),'position',[100 400 600 300]);
plot(wt')
title('Evolution of weights')
```