Programs to Bayesian statistics

November 17, 2020

Contents

1 Step response

First order regression model

$$y_t = a_1 y_{t-1}$$

Step response $\times$ *pulse response*

$a_1 < 0$ - *oscillations with the period*

T10simCont.sce - 1st order model

- $a = 0.9$, $b = [1, 0]$ - slow response;
- $a = 0.4$ - quick response

Second order regression model

Stability - roots of characteristic equation are within unit circle

$$y_t + A_1 y_{t-1} + A_2 y_{t-2} = 0 \rightarrow z^2 + A_1 z + A_2 = 0 \rightarrow r_1, r_2$$

The response depends on characteristic equation

– two real roots, – one double root, – two complex roots

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} \;\rightarrow\; y_t - a_1 y_{t-1} - a_2 y_{t-2} = 0$$

$$z^2 - a_1 z - a_0 = (z - r_1)(z - r_2) = z^2 - (r_1 + r_2) z + r_1 r_2$$

$$\rightarrow \;\; a_1 = r_1 + r_2; \;\; a_2 = -r_1 r_2$$

The output steady-state is $(y_t = y_{t-1} = y_{t-2} = y)$

$$y = a_1 y + a_2 y + u \;\rightarrow\; y - a_1 y - a_2 y = u$$

$$\frac{y}{u} = \frac{1}{1 - a_1 - a_2}$$

Pulse response - $u_1 = 0$; $y_0 = 1$

Step response - $u_t = 1$; init. cond. $= 0$

T11simCont.sce - 2nd order model

- $r_1 = 0.3; r_2 = 0.2; \rightarrow a = [0.7, -0.06]$ - standard response
- $r_1 = r_2 = 0.95$ - double root (delay at the beginning)
- $r_1 = 0.7 + 0.3i$ - damped oscillations (re $\pm$, im $\pm$)
- $r_1 = 0.7 + 0.8i$ - unstable

2 Simulation

Regression model

$$y_t = b_0 u_t + a_1 y_{t-1} + b_1 u_{t-1} + \cdots + a_n y_{t-n} + b_n u_{t-n} + k + e_t$$

$$e_t = \text{sd*rand(1,1,'n')}$$

Parameters a - see above

Parameters b - nothing much

Input signal + noise

T12simCont.sce - 2nd order model

- $a = [.4, .5], \quad b = [1, .2, -.5]$ - a general noisy system
- $a = [.4, .7], \quad b = [1, .2, -.5]$ - unstable system

Discrete model

$$f\left(y_t|u_t, y_{t-1}, \Theta\right)$$

$$k = \text{ny*(u(t)-1)+y(t-1)}$$

$$y_t = \text{sum(rand(1,1,'u')>cumsum(th(k,:)))+1;}$$

Formulate function of a deterministic model and realize it

T13simCont.sce - controlled coin with memory

- $\Theta = [.2, .8;\ .6, .4;\ .9, .1;\ .3, .7]$ - general model with uncertainty
- $\Theta = [.002, .998;\ .006, .994;\ .999, .001;\ .993, .007]$ - deterministic model

3 Estimation

Regression model - LS estimation

$$\hat{\theta}_t = \left(X'X\right)^{-1} X'Y$$

T21estCont_LS.sce - LS estimation

excited by sine function only - bad (has little information): sine + small noise - good

Regression model - Bayesian estimation

$$V_t = V_{t-1} + \Psi_t \Psi_t^{'};\quad \kappa_t = \kappa_{t-1} + 1$$

$\Psi_t = [y_t, u_t, y_{t-1}u_{t-1} \cdots, 1]' = \left[y_t, \psi_t'\right]$

$$V_t = \begin{bmatrix} V_y & V'_{y\psi} \\ V_{y\psi} & V_\psi \end{bmatrix}, \quad \hat{\theta}_t = V_\psi^{-1} V_{y\psi}$$

T22estCont_B.sce - Bayesian on-line estimation with statistic update

- rewrite with a single time loop - T22estCont_B1.sce

T22estCont_B2.sce

- model for simulation differs from that for estimation (structure mismatch)
- choice of input signal - influence on estimation

T22estCont_B3.sce

- model mismatch (simulation is fixed, estimation is optional)
- choice of input signal

T22estCont_B4.sce - real data from Strahov (varying model constant; better with daily course)

T22estCont_B5.sce - solution with daily course (estimated as quadratic regression in parabola.sce)

Discrete model

$$V_{y_t|[u_t,y_{t-1}];t} = V_{y_t|[u_t,y_{t-1}];t-1} + 1$$

$\cdots$ only one entry of the statistics V is incremented.

$$\hat{\Theta}_t = \aleph_r\left(V_t\right)$$

$\cdots$ $\aleph_r\left(\cdot\right)$ means normalization over rows

T23estDisc.sce

- model with various uncertainty (precision of results)
- deterministic model with given functionality (e.g. y follows u)

4 Prediction

Point prediction

$$d(0) = \{y_0, y_{-1}, y_{-2}\}$$

$$\hat{y}_1 = a_1 y_0 + a_2 y_{-1} + a_3 y_{-2}$$

$$\hat{y}_2 = a_1 \hat{y}_1 + a_2 y_0 + a_3 y_{-1}$$

$$\hat{y}_3 = a_1 \hat{y}_2 + a_2 \hat{y}_1 + a_3 y_0$$

$$\hat{y}_4 = a_1 \hat{y}_3 + a_2 \hat{y}_2 + a_3 \hat{y}_1$$

$$\hat{y}_4 = a_1 \hat{y}_4 + a_2 \hat{y}_3 + a_3 \hat{y}_2$$

etc.

Regression model

T31preCont.sce - known parameters (very accurate even for long period)

- error is given only by noise (for zero noise is precise)

T32preCont_Adapt.sce - on-line estimation (gradual improvement)

- compare SE at the beginning and the end
- change length of prediction
- set u_t without noise

T32preCont_Adapt2.sce - on-line estimation / model mismatch

T32preCont_Adapt3.sce - real data (Strahov tunnel)

- variable constant - better with daily curse (using polynom)

T32preCont_Adapt4.sce - real data (using daily course)

- uses polynom of 11th order - not ideal (better spline)
- spline approximation (of 3rd order polynomials)

Discrete model

T33preCat_Off.sce - known parametrs

- parameters - deterministic $\times$ with uncertainty

T34preCat_OffEst.sce - adaptive (off-line - first estimation and then prediction)

- uncertain model - bad
- almost deterministic - good

T35preCat_OnEst.sce - adaptive (on-line - in a single time loop)

- deterministic $\times$ uncertain model
- random $\times$ constant input
- various length of prediction (0, 5, 15)

5 Filtration

```
[xp(:,t),Rx,yp(t)]=Kalman(xp(:,t-1),yt(t),ut(t),M,N,[],A,B,[],Rw,Rv,Rx);
```

State estimation

T46statEst_KF.sce - Kalman filter

- derive the state-space model parameters from a regression model
- compare th properties of the simulated data with those of the regression model
  stable × unstable; monotonous × oscillating etc.
- judge the quality of estimation for individual cases

Noise filtration

T47statEst_Noise.sce - Kalman filter used as noise filter

- various setting of r_v and r_w

6 Control

$R = 0;$

for $t = N \cdots 1$

$T = R + \Omega$
$A = N'TN$
$B = N'UM$
$C = M'UM$
$S_t = A^{-1}B$
$R = C - S_t' A S_t$

end

for $t = 1 : N$

$u_t = -S_t x_{t-1}$
$y_t = \text{system}\,(u_t)$

end

T50ctrlMinVar.sce - minimum variance control

- minimal $\times$ nonminimal phase systems

T51ctrlX.sce - optimal control with regression model

- set various ω and λ and check control behavior
- change the setpoint

T52ctrlXEst.sce - adaptive sub-optimal control with regression model

- $n_i = 1, 5, 10, 20$ - initial learning
- $R = 0$ before the loop $\times$ in the loop
- change n_h
- change ω and λ
- all with $b_1 = -0.4$ and $b_1 = -1.4$ (nonminimum phase)

T53ctrlDisc.sce - optimal control with discrete model

- change model parameters: with $\times$ without uncertatinty
- change loss function with some interpretation and check it

7 Functions

v2thN.sci - parameter point estimation for regression model

```
function [th,s2]=v2thN(v,m)
  // [th,s2]=v2thn(v,m)   computation of par. point estimates
  //                      from normalized information matrix
  // v    information matrix
  // m    dimension of y
  // th   regression coefficients
  // s2   noise covariance estimate

  if argn(2)<2    // check for number of input arguments
    m=1;
  end

  // partitioning of information matrix
  vy=v(1:m,1:m);
  vyf=v(m+1:$,1:m);
  vf=v(m+1:$,m+1:$);

  // computation of point estimates
  th=inv(vf+1e-12*eye(vf))*vyf;
  s2=(vy-vyf'*th);
endfunction
```

Kalman.sci - Kalman filter procedure

```
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
```

psi2row.sci - regression vector from number of row of model matrix

```
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
```

row2psi.sci - row of model matrix from regression vector

```
function x=row2psi(i,b)
  // x=row2psi(i,b)   generates a discrete rgegression vector
  //                  with a base b that corresponds to the
  //                  i-th row of a table of the discrete model
  // 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)

  if isscalar(b),
    n=fix(log(i+1)/(log(b)))+1;
    b=b*ones(1,n);
  else
    n=length(b);
  end
  if i>prod(b)
    disp('ERROR: The row number is too big')
    return
  end
  i=i-1;
  for j=1:n
    i=i/b(n-j+1);
    x(n-j+1)=round((i-fix(i))*b(n-j+1)+1);
    i=fix(i);
```

```
  end
  x=x(:)';
endfunction
```

filtw.sci - noise filter using averaging

```
function xf=filtw(x,nf)
  // xf=filtw(x,nf)
  // filtration by averaging over a window
  // xf    filtered signal
  // x     signal to be filtered
  // nf    radius of the window (+ central, - back)
  //
  n=max(size(x));
  xf=x;
  if nf==0, return, end
  if nf>0
    for i=(nf+1):(n-nf-1)
      xf(i)=mean(x((i-nf):(i+nf)));
    end
  else
    nf=-nf;
    for i=(nf+1):n
      xf(i)=mean(x((i-nf):i));
    end
  end
endfunction
```

filts.sci - noise filter using spline approximation

```
function f=filts(x,n,na,nb,nc)
  // f=filts(x,n,na,nb,nc)
  // signal filtration (smoothing) by 3rd order splines
```

```
  // x   signal
  // n   length of polynomial
  // na  number of fictitious data for contiunity
  // nb  number of fictitious data for for smooth
  //     connecting (1st derivative)
  // nc  number of fictitious data for overal
  //     smoothness (2nd derivative)
  //
  b=(x(2)-x(1))/n;
  a=x(1)+b;                    // leftmost initial condition
  nn=fix(max(size(x))/n);      // number of polynomials
  f=[];
  for i=1:nn                   // loop over polynomials
    ii=(1:n)+(i-1)*n;          // time points of single polynomial
    f=[f filt3(x(ii),a,na,b,nb,nc)]; // new polynomial
    b=(f($)-f($-1))/n;         // condition for smotthness
    a=f($)+b;                  // condition for continuity
  end
  if max(size(x))>max(size(f)) // last piece of the spline
    f=[f filt3(x((max(size(f))+1):$),a,na,b,nb,nc)];
  end
endfunction
```

filt3.sci - construction of 3rd order polynomial (support for filts)

```
function [f,th]=filt3(x,vL,nv,dL,nd,p2,I,ni)
  // f=filt3(x,vL,nv,dL,nd,p2,I,ni)
  // approximation by the 3rd order polynomial
  // x   signal to be approximated
  // vL  value on the left (border condition)
  // nv  number of data for fixing the left border
  // dL  derivative on the left (border condition)
  // nd  number of data for fixing the equality of derivatives
```

```
  // p2  relative number of data for fixing zero second deriv.
  // I   integral of the approximative polynomial
  // ni  number of data for fixing the pre-scribed integral
  //
  if argn(2)<7, I=0; ni=0; end

  n=max(size(x));                          // data length
  d=[];
  for i=(1:n)-1                            // data generation
    d=[d [x(i+1); 1; i; i^2; i^3]];        // fitting data
    d=[d [0; 0; 0; 2; 6*i]*p2];            // 2nd derivative cond.
  end
  d=[d [vL; 1; 0; 0; 0]*nv];               // cont. condidion
  d=[d [dL; 0; 1; 0; 0]*nd];               // derivatives on left
  d=[d [I; n-1; (n-1)^2/2; (n-1)^3/3; i^4/4]*ni]; // integral
  V=d*d';                                  // information matrix
  V=V+eye(V)*1e-5;                         // regularization
  th=v2thN(V/n,1);                         // point estimates
  f=zeros(1,n);
  for i=(1:n)-1                            // polynomial generation
    f(i+1)=th'*[1; i; i^2; i^3];
  end
endfunction
```