Bibliografia

[1] ANDERSON, B; MOORE, J. **Optimal Filtering**. Englewood Cliffs, NJ: Prentice Hall, 1979.

[2] CARLIN, B; POLSON, N; STOFFER, D. **A Monte Carlo approach to nonnormal and nonlinear state-space modelling**. J. American Statistical Association, 87:493–500, 1992.

[3] CARTER, C; KOHN, R. **On Gibbs sampling for state space models**. Biometrika, 81:541–553, 1994.

[4] CARGNONI, C; MÜLLER, P; WEST, M. **Bayesian Forescasting of multinomial time series through conditionally Gaussian dynamic models**. J. American Statistical Association, 92:640–647, 1997.

[5] DOORNIK, J. **Ox 3.O: An Object-Oriented Matrix Programming Languaje**. Timberlake Consultants Ltd., 4st edition, 2001.

[6] DOUCET, A; FREITAS, J; GORDON, N. **Sequential Monte Carlo in Practice**. New York:Springer-Verlag, 2001.

[7] DURBIN, J; KOOPMAN, S. **Time Series Analysis by State Space Methods**. Oxford University Press, 2001.

[8] DURBIN, J; KOOPMAN, S. **Monte Carlo maximum likelihood estimation for non-Gaussian state space models**. Biometrika, 84:669–684, 1997.

[9] DURBIN, J; KOOPMAN, S. **Time Series analysis of non-Gaussian observations based on state space models from both classical and Bayesian perspectives**. J. Royal Statistical Society B, 62:3–56, 2000.

[10] DURBIN, J; KOOPMAN, S. **A simple and efficient simulation smoother for state space time series analysis**. Biometrika, 89:603–616, 2002.

[11] FAHRMEIR, L. **Conditional mode estimation by extended Kalman Filtering for multivariate dynamic generalised linear models**. J. American Statististical Association, 87:501–509, 1992.

[12] FERNANDES, C. **Non-Gaussian Structural Time Series Model**. PhD thesis, University of London, 1990.

[13] FLETCHER, R. **Practical Methods of Optimisation**. New York: John Wiley, 2nd edition, 1987.

[14] GAMERMAN, D. **Markov Change Monte Carlo: Stochastic Simulations for Bayesian Inference**. London:Chapman and Hall, 1997.

[15] GAMERMAN, D. **Markov chain Monte Carlo for dynamic generalised linear models**. Biometrika, 85:215–227, 1998.

[16] HARVEY, A; DURBIN, J. **The effects of seat belt legislation on British road casualties: A case study in estructural time series modelling, (with discussion)**. J. Royal Statistical Society A, 149:187–227, 1986.

[17] HARVEY, A. **Forescasting, Structural Time Series Models and the Kalman Filter**. Cambridge: Cambridge University Press., 1989.

[18] HARVEY, A; FERNANDES, C. **Time series models for count data or qualitative observations**. Business and Economic Statist., 7:407–17, 1989.

[19] HARVEY, A; SHEPHARD, N. **Structural time series models**. In Maddala, G.S., Raco, C.R. and Vinod, H.D. (eds.), Handbooks of Statistics. Amsterdam:Elsevier Science Publishers., 11:261–301, 1993.

[20] JOHNSON, N; KOTZ, S. **Distributions in statistic: discrete distribution**. Mifflin series in statistic, 1969.

[21] JONG, P DE; SHEPARD, N. **The simulation smoother for time series models**. Biometrika, 82:339–50, 1995.

[22] KALMAN, R. **A new approach to linear filtering and prediction problems**. J.Basic Engineering Transaction ASMA, 82:35–45, 1960.

[23] KOCHERLAKOTA, S; KOCHERLAKOTA, K. **Bivariate Discrete Distributions**. New York: Dekker, 1992.

[24] KOOPMAN, S. **Exact initial Kalman filtering and smoothing for non-stationary time series models**. J. American Statistical Association, 92:1630–38, 1997.

[25] KOOPMAN, S; SHEPHARD, N; DOORNIK, J. **Statistical algorithms for models in state space using SsfPack 2.2**. Econometric Journal, 2:113–166, 1999.

[26] KOOPMAN, S; SHEPHARD, N; DOORNIK, J. **Statistical algorithms for models in state space**. Não publicado, disponivel só através dos autores, 2002.

[27] KOOPMAN, S; HARVEY, A; DOORNIK, J; SHEPHARD, N. **Stamp 6.0: Structural Time Series Analyser, Modeller and Predictor**. London: Timberlake Consultants, 1st edition, 2000.

[28] PITT, M; SHEPARD, N. **Filtering via simulation: auxiliary particle filter. ,94,590-599.** J. American Statistical Association, 94:590–599, 1999.

[29] RIPLEY, B. **Stochastic Simulation**. New York: Wiley, 1987.

[30] ROSENBERG, B. **Random coefficients models: the analysis of a cross-section of time series by stochastically convergent parameter regression**. Annals of Economic and Social Meassurement, 2:399–428, 1973.

[31] SHEPARD, N. **Partial non-Gaussian state space**. Biometrika, 81:115–131, 1994.

[32] SHEPARD, N; PITT, M. **Likelihood analysis of non-Gaussian measurement time series**. Biometrika, 84:653–667, 1997.

[33] TANIZAKI, H. **Nonlinear Filters**. Springer, 2nd edition, 1996.

[34] WEST, M; HARRISON, P; MIGON, H. **Dynamic generalized linear models and Bayesian forescasting (with discussion)**. J. American Statistical Association, 80:73–97, 1985.

[35] WEST, M; HARRISON, P. **Bayesian Forecasting and Dynamic Models**. Springer-Verlag, 1997.

A
Apêndice

A.1
Modelo Poisson Bivariado proposto por Fernandes

Equação das observações

Suponha que existam duas séries de observações de dados de contagem, cada uma das séries seguindo uma distribuição de Poisson, isto é,

$$p(y_{it}|\theta_{it}) = \frac{\theta_{it}^{y_i t} \exp(-\theta_{it})}{y_{it}!}, \qquad i = 1, 2, \qquad t = 1, \ldots, n,$$

onde θ_{i1}, $i = 1, 2$ é o sinal.

A divisão nas taxas individuais θ_{it} é dada pela relação

$$\begin{aligned} \theta_{it} &= \pi_{it}\theta_t, \qquad \text{onde} \\ \pi_{1t} + \pi_{2t} &= 1, \qquad 0 < \pi_{it} < 1, \qquad i = 1, 2, \end{aligned}$$

onde θ_t é a taxa total, ainda não definida.

Seja o vetor $w_t = (y_{1t}, y_{2t})$. Posto que as séries são independentes, condicional ao conhecimento de suas respectivas taxas, então a equação das observações pode ser escrita como:

$$p(w_t|\theta_{1t}, \theta_{2t}) = \prod_{i=1}^{2} p_i(y_{it}|\theta_{it}),$$

onde $p_i(\cdot)$ denota a distribuição de probabilidade de y_{it}.

Considere agora a série agregada S_t, definida por $S_t = y_{1t} + y_{2t}$, $t = 1, \ldots, n$.

S_t é uma variável aleatória com distribuição Poisson de taxa θ_t, isto é,

$$p(S_t|\theta_t) = \frac{\theta_t^{S_t} \exp(-\theta)}{S_t!}.$$

Deste modo os π's podem ser interpretados como a divisão associada a cada série.

Pode-se mostrar que:

$$p(y_{1t}|S_t, \pi_{1t}) = \begin{pmatrix} S_t \\ y_{1t} \end{pmatrix} \pi_{1t}^{y_1 t}(1-\pi_{1t})^{S_t - y_{1t}},$$

ou seja $p(y_{1t}|S_t, \pi_{1t})$ é binomial.

A equação da observações pode ser escrita como:

$$p(w_t|S_t, \theta_t, \pi_{1t}) = p(S_t|\theta_t)p(y_{1t}|S_t, \pi_{1t}).$$

Deve-se agora considerar mecanismos estocásticos para evolução da taxa total θ_t e a divisão π_{1t}.

Predição do Estado

Seja $Y_t = \{Y_{1t}, Y_{2t}\}$, onde $Y_{it} = (y_{i1}, y_{i2}, \ldots, y_{it})$, $i = 1, 2$.

1. *Taxa Total:*

   Fernandes [12] propõe uma *priori* gama da forma:

   $$p(\theta_{t-1}|Y_{t-1}) \sim gamma(a_{t-1}, b_{t-1}).$$

   uma vez que S_t é Poisson, é possível mostrar que neste caso: $\theta_t|Y_{t-1} \sim gamma(a_{t/t-1}, b_{t/t-1})$. (A forma de obtenção de $a_{t/t-1}$ e $b_{t/t-1}$ é apresentada no capítulo 3 do trabalho de Fernandes.)

2. *Divisão individual:*

   Dada a restrição $\pi_{1t} + \pi_{2t} = 1$, só é necessário considerar a dinâmica de uma das componentes, seja esta $\pi \equiv \pi_{1t}$. Dado que $p(y_{1t}|S_t)$ é binomial, a *priori* considerada é:

   $$p(\pi_{t-1}|Y_{t-1}) \sim beta(c_{t-1}, d_{t-1}).$$

   Neste caso Fernandes [12] mostra que a distribuição a *posteriori* é $\pi_t|Y_{t-1} \sim beta(c_{t|t-1}, d_{t|t-1})$. (A forma de obter $c_{t|t-1}$ e $d_{t|t-1}$ é descrita no capítulo 4 do trabalho de Fernandes.)

Atualização do estado:

Como antes, os resultados obtidos estão no trabalho de Fernandes [12], o qual apresenta a forma de obtenção dos parâmetros considerados.

1. *Taxa total:*

   A *posteriori* para θ_t, $p(\theta_t|Y_t)$ é gama com parâmetros (a_t, b_t).

2. *Divisão individual:*

   A *posteriori* é beta com parâmetros (c_t, d_t).

Fernandes [12] aplica o seu modelo às séries de gols marcadas nos jogos Escócia e Inglaterra.

A.2
Programa utilizado na tese

A seguir é listado o programa Ox 3.0 considerado na implementacao da tese.

```
/*
  Programa que faz a estimacao do estado para os dados considerados na tese
*/

#include <oxstd.h> #include <oxfloat.h> #include <oxdraw.h>
#include <oxprob.h> #import <maximize> #import<packages/ssfpack.h>

static decl T,M,z_1t,z_2t,lambda,my,dlik,ir,aleat,mphi,mIndex,
mOmega; static decl mga,par,delta,theta_final, theta_otimo;

// funcao que calcula a funcao de probabilidade do modelo proposto
densidad(const T,const lambda,const my,const delta) {
    decl i,j,min=minc(my),P=zeros(1,T);
   for(i=0;i<T;++i)
    {
      if (min[i]<0)
      {
      P[i]=0;
      }
      else
      {
        for(j=0;j<=min[i];++j)
        {
          P[i] +=exp(-lambda-delta[0][i]-delta[1][i])
          *lambda^j/gammafact(j+1)*delta[0][i]^(my[0][i]-j)
```

```
          /gammafact(my[0][i]-j+1)*
          delta[1][i]^(my[1][i]-j)/gammafact(my[1][i]-j+1);
        }
      }
    }
    return P;
}
//     Funcao que permite calcular o yt e Ht em cada iteracao na busca do MGA
obtem_var_covar(const my,const delta) {
    decl pp2,Ht=<>,a1=<1;0>,my1_1,py1_1,my1_2,py1_2;
    decl a2=<0;1>,my2_1,py2_1,my2_2,py2_2,p;
    decl b12=<1;1>,my1y2,py1y2;
    decl m11,m12,m22,my2=<>,c2t=zeros(2,T);
    decl i,inv=zeros(2,2),H_tparcial=zeros(2,2);
    p=densidad(T,lambda,my,delta);// p(y1,y2)
    pp2=p.^2; // p^2
    my1_1=my-a1;
    py1_1=densidad(T,lambda,my1_1,delta);// p(y1-1,y2)
    my1_2=my1_1-a1;
    py1_2=densidad(T,lambda,my1_2,delta);//p(y1-2,y2)
    my2_1=my-a2;
    py2_1=densidad(T,lambda,my2_1,delta);//p(y1,y2-1)
    my2_2=my2_1-a2;
    py2_2=densidad(T,lambda,my2_2,delta);//p(y1,y2-2)
    b12=<1;1>;
    my1y2=my-b12;
    py1y2=densidad(T,lambda,my1y2,delta);//p(y1-1,y2-1)
    m11=(-py1_2.*p+py1_1.^2)./pp2;
    m12=(-py1y2.*p+py2_1.*py1_1)./pp2 ;
    m22=(-py2_2.*p+py2_1.^2)./pp2;
    c2t[0][]=((py1_1-p)./p).*delta[0][];
    c2t[1][]=((py2_1-p)./p).*delta[1][];
    for(i=0;i<T;++i)
    {
     inv[0][0]=m11[][i]*(delta[0][i]^2)-c2t[0][i];
     inv[1][1]=m22[][i]*(delta[1][i]^2)-c2t[1][i];
     decl amval;
     eigen(inv, &amval);
     decl min_eing = min(amval);
```

```
     if(min_eing==.NaN )
     {
          H_tparcial = unit(2);
     }
     else
     {
     if(min_eing<=0.00)
     {
          H_tparcial = unit(2);
     }
     else
     {
         H_tparcial=invert(inv);// Ht for each t
     }
     }
     decl theta_parcial=log(delta[][i]+.0001); // cuidado con el lambda
     decl my2_parcial=theta_parcial+H_tparcial*c2t[][i];// yt from the AGM
     Ht=Ht~vec(H_tparcial);
     my2=my2~ my2_parcial;
     }
//  matriz que contiene   yt e  Ht para el MGA
    mga= my2|constant(exp(par[0][0]),1,T)|Ht[0:1][]|Ht[2:3][];
    return mga;
}

// funcao para achar o MGA
SsfApproxModel() {
    decl s_vTheta=<>,i,dconv,ms,y_antigo,y_novo,Delta;
    ms=SsfCondDensEx(DS_SMO,mga[0:1][],mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    for (i=0;  i<300 ;i++)
    {
    y_antigo=mga[0:1][];
    s_vTheta=mga[0:1][]-ms[1:2][];// evaluates initial theta_t
    Delta=exp(s_vTheta)-.0001;// new delta_t
    obtem_var_covar(my,Delta);// news H_t and Y_t
    ms=SsfCondDensEx(DS_SMO,mga[0:1][],mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    y_novo=mga[0:1][];
    dconv=max(fabs((y_antigo-y_novo)));
    if(dconv < 10^-5) break
```

```
    }
}
// obtem a funcao de probabilade do MGA
logdens_gauss(const err,const mga) {   decl
i,H_tg,G=zeros(1,T-1),dret_g;
    for(i=0;i<T-1;++i)
    {
     H_tg = mga[3:4][i] ~mga[5:6][i];
      G[][i]=-0.5* err[][i]'*invert(H_tg)* err[][i];
     }
    dret_g=sumr(G);
    return dret_g;
}
//   Algoritmo para fazer Simulation Smoothing (considering DK 2001)
// e obter os pesos. sim_smoo(const chol,const teta_hat,const
dlhat) {
    decl i,dif,alpha_plus;
    decl pert_obs,pert_est;
    decl theta_sim,delta_sim,w=zeros(2,1),y_plus,teta_plus;
    pert_obs=chol*rann(2*(T),1);//gera as pert. das medidas
    pert_est=sqrt(exp(par[0][0]))*rann(1,T) ; // gera as pert. do estado
    alpha_plus=zeros(1,T+1);
    y_plus=zeros(2,T);
    teta_plus=zeros(2,T);
    for(i=0;i<T;++i)
    {
     alpha_plus[i+1] = alpha_plus[i]+ pert_est[i];
      teta_plus[0][i] =mphi[1][0] *alpha_plus[i] ;
     teta_plus[1][i]  = mphi[2][0]*alpha_plus[i] ;
     y_plus[0][i]     = teta_plus[0][i]+pert_obs[2*i];
     y_plus[1][i]     = teta_plus[1][i]+pert_obs[2*i+1];
    }

    decl y_star=mga[0:1][]-y_plus;
    decl teta_hat_sim;
    decl teta_hat_sim0=SsfMomentEstEx(ST_SMO,&teta_hat_sim,y_star,
    mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    theta_sim=teta_hat_sim[1:2][]+teta_plus;
    decl theta_sim_ant=2*teta_hat-theta_sim;
```

```
    delta_sim=exp(theta_sim);
    dif=sumr(log(densidad(T,lambda,my,delta_sim)))-
    logdens_gauss((mga[0:1][]-theta_sim),mga);
    w[0]=exp(dif-dlhat);
    decl delta_sim_ant=exp(theta_sim_ant);
    dif=sumr(log(densidad(T,lambda,my,delta_sim_ant)))-
    logdens_gauss((mga[0:1][]-theta_sim_ant),mga);
    w[1]=exp(dif-dlhat);
    return w;
    }
// funcao usada para achar a estimacao do estado (apos o processo de max.)
sim_smoo_nova(const chol,const teta_hat,const dlhat) {
    decl i,dif,alpha_plus;
    decl pert_obs,pert_est;
    decl theta_sim,delta_sim,w=zeros(2,1),y_plus,teta_plus;
    pert_obs=chol*rann(2*(T),1);//gera as pert. das medidas
    pert_est=sqrt(exp(par[0]))*rann(1,T);// gera as pert. do estado
    alpha_plus=zeros(1,T+1);
    y_plus=zeros(2,T);
    teta_plus=zeros(2,T);
    for(i=0;i<T;++i)
    {
     alpha_plus[i+1] = alpha_plus[i]+ pert_est[i];
      teta_plus[0][i] = z_1t*alpha_plus[i] ;
     teta_plus[1][i]  = z_2t*alpha_plus[i] ;
     y_plus[0][i]     = teta_plus[0][i]+pert_obs[2*i];
     y_plus[1][i]     = teta_plus[1][i]+pert_obs[2*i+1];
    }

    decl y_star=mga[0:1][]-y_plus;
    decl teta_hat_sim;
    decl teta_hat_sim0=SsfMomentEstEx(ST_SMO,&teta_hat_sim,
    y_star,mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    theta_sim=teta_hat_sim[1:2][]+teta_plus;
    decl theta_sim_ant=2*teta_hat-theta_sim;
    delta_sim=exp(theta_sim);
    theta_final =   delta_sim;
    dif=sumr(log(densidad(T,lambda,my,delta_sim)))-
    logdens_gauss((mga[0:1][]-theta_sim),mga);
```

```
    w[0]=exp(dif-dlhat);
    decl delta_sim_ant=exp(theta_sim_ant);
    dif=sumr(log(densidad(T,lambda,my,delta_sim_ant)))-
    logdens_gauss((mga[0:1][]-theta_sim_ant),mga);
    w[1]=exp(dif-dlhat);
    theta_otimo =  theta_sim|theta_sim_ant;
    return w;
    }
//  Verossimilhanca por Monte Carlo
MClik() {

    decl i,Wgt=zeros(2,M),dg,d,mn,vr,delta_hat,teta_hat,dlhat;
    SsfApproxModel();
    SsfLikEx(&dg,&d,mga[0:1][],mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    ranseed(5);
    decl teta_hat1;
    decl teta_hat0=SsfMomentEstEx(ST_SMO,&teta_hat,mga[0:1][],
    mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    delta_hat=exp(teta_hat[1:2][]);
    dlhat=sumr(log(densidad(T,lambda,my,delta_hat)))-
    logdens_gauss((mga[0:1][]-teta_hat[1:2][]),mga);
    decl H_tg,diago=mga[3:4][0] ~mga[5:6][0];
    for(i=1;i<T;++i)
    {
     H_tg = mga[3:4][i] ~mga[5:6][i];
     diago =diagcat(diago,H_tg);
     }
    decl chol=choleski(diago);
    for(i=0;i<M;i++)
    {
      Wgt[][i]=sim_smoo(chol,teta_hat[1:2][],dlhat);
    }
    decl pesos=Wgt[0][:]~Wgt[1][:];
    mn=meanr(pesos);
    vr=varr(pesos);
    return double (dg+dlhat+log(mn)+(0.5*vr/(columns(pesos)*sqr(mn))));
}

Likelihood(const vP,  pdlik,const pvsco, const pmHes) {
```

```
    mOmega[0][0]=exp(vP[0][0]);
    mga[2][]=   exp(vP[0][0]);
    mphi[2][0]=par[1][0];
    pdlik[0]=4*MClik()/(T) ;
    return 1;
}

MaxLik() {
    MaxControl(300,1,1);
//  MaxControlEps(10e-4,10e-4);
    MaxControlEps(1,1);
    ir=MaxBFGS(Likelihood,&par,&dlik,0,1);
    print("par",mOmega[0][0]|mphi[1:][]);
}

mean() {
    decl i,W=zeros(2,M),dg,d,mn,vr,delta_hat,teta_hat,dlhat;
    ranseed(5);
    decl teta_hat0=SsfMomentEstEx(ST_SMO,&teta_hat,mga[0:1][],
    mphi,mOmega,<>,<>,<>,mIndex,<>,mga);
    delta_hat=exp(teta_hat[1:2][]);
    dlhat=sumr(log(densidad(T,lambda,my,delta_hat)))-
    logdens_gauss((mga[0:1][]-teta_hat[1:2][]),mga);
    decl H_tg,diago=mga[3:4][0] ~mga[5:6][0];
    for(i=1;i<T;++i)
    {
     H_tg = mga[3:4][i] ~mga[5:6][i];
     diago =diagcat(diago,H_tg);
     }
    decl chol=choleski(diago);
    decl estado_est=zeros(1,T);
    decl saida_sim_smoo=zeros(1,T);
    for(i=0;i<M;i++)
    {
       W[][i]=sim_smoo_nova(chol,teta_hat[1:2][],dlhat);
      estado_est+=W[0][i]*theta_otimo[0][]+W[1][i]*theta_otimo[2][];
     }
    decl m=sumr(sumc(W));
    println("m",m);
```

```
    estado_est /=m;
    estado_est=estado_est;
    println("estado_est final ",estado_est);
    DrawTMatrix(0,my[0][]|exp(estado_est),{"y_1","estado estimado"},1,1,1);
    DrawTitle(0, "Modelo de Nível Local");
    ShowDrawWindow();
}

main() {
//  generates BVP observations
    M=200;  // numero de pesos considerados
    lambda=2;
    par=<-3.65;.8>;//valores inicias
    mphi = <1;1;1>;
    mOmega = <1,0,0;0,1,0;0,0,1>;
    mIndex =<2,-1,-1;-1,3,5;-1,4,6>;
    decl dados=loadmat("neonit.in7")';// leitura dos dados
    my = dados[][:95];
    T = columns(my);
    println("T",T);
    mphi[2]=par[1];
    delta=my;
    obtem_var_covar(my,delta);// obtencao de valores inicias
    MaxLik();
    mean();
}
```