Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Estatística

Notas em Computação Científica e Estatística - 2º semestre de 2014

Relatório Técnico RTE 03/14

F. R. B. Cruz (org.), G. L. Oliveira, J. C. Rocha, J. A. Almeida, P. Cerqueira Jr. e T. P. Martins Junior

Relatório Técnico
Série Ensino

Sumário

Prefácio **v**

1 Uma Aplicação do Modelo Partição Produto: O Algoritmo de Crowley (1997) para Gerar as Partições **1**
1.1 Introdução . . . 1
1.2 Metodologia . . . 2
1.2.1 O Modelo Partição Produto (MPP): Uma Visão Geral . . . 2
1.2.2 A Proposta de Crowley (1997) para Gerar as Partições . . . 3
1.3 Especificação e Estruturação Lógica do Programa . . . 4
1.3.1 A Classe Proposta (*arquivo .hpp*) . . . 5
1.4 Resultados e Conclusões . . . 6
A. Implementações (extensões do *arquivo .hpp*) . . . 8
B. Programa principal (*main*, *arquivo .cpp*) . . . 11

2 Método de Reamostragem *Bootstrap* via C++ **13**
2.1 Introdução . . . 13
2.1.1 Objetivos . . . 14
2.1.2 Organização . . . 14
2.2 Metodologia . . . 14
2.2.1 O Estimador *Bootstrap* do Erro Padrão . . . 14
2.2.2 O Estimador do Intervalo de Confiança Percentílico *Bootstrap* para a Média . . . 15
2.3 Resultados Experimentais . . . 17
2.4 Conclusões e Observações Finais . . . 17

3 Modelos Dinâmicos Lineares Generalizados: Implementação para a Distribuição de Poisson **19**
3.1 Introdução . . . 19
3.2 Modelos Lineares Generalizados Dinâmicos . . . 20
3.2.1 Definição . . . 20
3.2.2 Atualização . . . 21
3.2.3 Distribuição de Poisson . . . 23
3.3 Implementação . . . 24
3.3.1 Algoritmo . . . 24

3.3.2 Classe 24
3.4 Resultados 25
3.5 Considerações Finais 27
A. Implementação da Classe Proposta 28

4 Abordagem Bayesiana Dinâmica para o Ajuste do Modelo Exponencial por Partes 33
4.1 Introdução 33
4.2 Objetivos e Justificativa 34
4.3 Estimação via *Linear Bayes* 35
4.3.1 O Modelo 35
4.3.2 Inferência dos Parâmetros via Análise Sequencial 36
4.3.3 Atualização *on Line* 37
4.3.4 Atualização via Distribuição Suavizada 37
4.4 Resultados Pretendidos 38
4.5 Resultados Parciais 38
4.6 Resultados Esperados 38
4.7 Conclusões Parciais 39
A. Códigos 41

5 Simulação de Rebanho de Bovinos de Corte 45
5.1 Introdução 45
5.1.1 Objetivos 46
5.1.2 Organização 46
5.2 Materiais e Métodos 46
5.2.1 Entradas 46
5.2.2 Processamento 47
5.2.3 Saída 47
5.2.4 Classes Propostas 47
5.3 Resultados e Discussão 48
5.4 Conclusões e Observações Finais 50
A. Exemplo de Saída 52

Índice Remissivo 53

Prefácio

No decorrer do segundo semestre de 2014 foi lecionada a disciplina *Computação Científica e Estatística I*, oferecida pelo Programa de Pós-Graduação em Estatística do Departamento de Estatística da Universidade Federal de Minas Gerais. Direcionado a estudantes de pós-graduação em estatística ou área afins, a disciplina teve como objetivo principal fornecer uma introdução a vários aspectos da computação incluindo:

- uso de ferramentas para edição, compilação e correção de programas;
- metodologias modernas de programação (modularização, projeto orientado a objetos);
- projeto e implementação de estruturas de dados e algoritmos, bem como sua análise.

Ao final da disciplina, frente a um problema de pesquisa das suas respectivas áreas de interesse, os estudantes deveriam decidir qual é o seu tipo, em termos computacionais, saber onde procurar por ajuda e conhecer os termos mais usuais. Deveriam também ler artigos em áreas tais como análise numérica, simulação, computação científica, computação estatística ou estatística computacional e, finalmente, organizar e escrever programas de médio porte ($\approx$ 5-10 páginas), com a seleção de abstrações, estruturas de dados e algoritmos apropriados, bem como com o uso de programas existentes.

Apresenta-se a seguir uma seleção dos trabalhos que foram desenvolvidos no decorrer da disciplina. Embora sejam todos trabalhos em estágios iniciais, eles abordam temas atuais e instigantes, com grande potencial para futuros desenvolvimentos teóricos e práticos na estatística.

Finalmente, não poderia deixar de ser registrado aqui um agradecimento especial ao prof. Enrico Colosimo, pelas oportunidades oferecidas e pelo estímulo ao desenvolvimento de métodos computacionais no Departamento.

Tenham todos uma boa leitura e façam excelente proveito!

Frederico R. B. Cruz
Organizador

Uma Aplicação do Modelo Partição Produto: O Algoritmo de Crowley (1997) para Gerar as Partições

Guilherme Lopes de Oliveira
Departamento de Estatística - ICEx - UFMG
31270-901 - Belo Horizonte - MG
E-mail: guilopes2110@gmail.com

Resumo

Neste trabalho encontra-se a finalização do projeto de programação proposto para a disciplina Computação Científica e Estatística I. É apresentada uma contextualização e definições para o modelo partição produto (MPP) no contexto espacial a ser abordado aqui. Além disto, apresenta-se a classe proposta em C++ com algumas explicações da estrutura do algoritmo considerado e os resultados de sua implementação.

1.1 Introdução

Em Estatística Espacial diferentes métodos e modelos são utilizados para descrever como as taxas de um ou mais atributos se comportam em uma determinada região ou período de tempo. São diversas as áreas do conhecimento em que a análise de dados espaciais tem contribuído para uma melhor compreensão de fenômenos tais como saúde, ecologia, agronomia, geologia, sociologia, entre outras.

De fato, o estudo da distribuição espacial da incidência de doenças ou de outros acontecimentos relacionados à saúde é uma área com crescente relevância e grande utilização dentre os pesquisadores de epidemiologia e saúde pública, constituindo uma importante ferramenta de prevenção e construção de indicadores de saúde.

Neste contexto epidemiológico, um objetivo comum é estimar o risco de um indivíduo contrair uma doença ou mesmo falecer devido a algum fator. Geralmente, este risco é estimado através de alguma da taxa ou proporção. Estas taxas são, em geral, representadas em mapas para que se possa identificar tendências espaciais (ou temporais) e/ou fatores de risco ligados às doenças.

O desenvolvimento recente desta área de investigação se deve em grande parte aos avanços da computação. Tais avanços facilitaram um maior e melhor acesso aos sistemas de informação geográfica, além de facilitar o tratamento de modelos estatísticos complicados. Destacam-se também os avanços metodológicos na área da estatística, sobretudo os métodos bayesianos empíricos e hierárquicos. À propósito, os modelos bayesianos para mapeamento de doenças permitem, por exemplo, a obtenção de estimadores estáveis para as taxas de mortalidade em pequenas áreas, usando a informação de todas as sub-regiões para inferir sobre a taxa de mortalidade de cada sub-região.

Usando a abordagem bayesiana, Oliveira & Loschi (2013) apresentaram um mapeamento da mortalidade neonatal precoce em Minas Gerais baseado nos riscos relativos (RR) estimados usando um modelo Poisson censurado com efeitos aleatórios espaciais (Bailey et al. 2005). No geral, os resultados obtidos com os critérios de censura sugeridos pelos autores foram satisfatórios e superiores àqueles obtidos com o modelo não-censurado, embora sejam muito influenciados pelo critério de censura adotado.

Um método potencial para aprimorar os resultados de Oliveira & Loschi (2013) é o uso do modelo partição produto (MPP). O MPP foi introduzido, em sua forma mais geral, por Hartigan (1990) e desde então tem sido amplamente utilizado, por exemplo, em problemas que envolvem a investigação de pontos de mudança, observações atípicas, dentre outras.

Na sua forma geral, o MPP considera todas as partições possíveis do conjunto de dados levando em conta tanto blocos contíguos quanto não-contíguos. Barry & Hartigan (1992) particularizam o MPP para a situação em que os dados são sequencialmente observados e apenas blocos contíguos são permitidos. Neste trabalho eles introduzem uma versão paramétrica para o MPP em que os blocos presentes na partição são induzidos pela igualdade dos parâmetros que indexam as distribuições das observações. Numa tentativa de aprimorar o trabalho de Oliveira & Loschi (2013), a ideia é estender e aplicar o MPP no contexto de blocos espaciais não necessariamente contíguos. Para isto a maior dificuldade está na obtenção das partições espaciais.

O MPP é também utilizado por Crowley (1997), que propõe um algoritmo genérico para gerar as partições pertinentes ao modelo. Para estender o MPP ao contexto de blocos espaciais, implementa-se aqui o algoritmo de Crowley (1997). Será utilizada a programação orientada por objetos em linguagem C++ (Buzzy-Ferraris 1993).

1.2 Metodologia

1.2.1 O Modelo Partição Produto (MPP): Uma Visão Geral

Considere uma região dividida em n áreas. Para cada área A_i, $i = 1, \ldots, n$, observa-se o número de casos de uma doença, X_i. Faça θ_i denotar o risco relativo da doença nesta área e $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$. Além disso, considere que o número esperado de casos em cada área é E_i e que $X_i \mid \theta_i \sim \text{Poisson}(\theta_i E_i)$ independentes $\forall\ i$. Nosso interesse é estimar $\boldsymbol{\theta}$.

Para o MPP (Barry & Hartigan 1992), assumimos que existe uma partição da região em b componentes (sub-regiões, *clusters*), $\rho = \{S_1, \ldots, S_b\}$, $b \leq n$, tal que cada área A_i pertence

a exatamente uma das componentes e que o risco relativo (RR) é constante para as áreas que pertencem à mesma componente. Esta partição determina a existência de $\theta_{S_1}, \ldots, \theta_{S_b}$ tal que

1. $\theta_{S_k} = \theta_j,\ \forall j\ \in\ S_k,\ k = 1, ..., b$ são iid;

2. $\theta_{S_1}, ..., \theta_{S_b}$ são independentes com $\theta_{S_k} \sim \pi(\theta_{S_k})$;

1.2.2 A Proposta de Crowley (1997) para Gerar as Partições

O Algoritmo

No algoritmo apresentado no Quadro 1, vê-se a estrutura do programa a ser desenvolvido em C++.

algoritmo
leia dados
inicialize partição
repita (para cada região)
defina uma partição auxiliar trocando o elemento i de *cluster*
calcule número de *clusters* na partição auxiliar
calcule número de elementos em cada *cluster* da partição auxiliar
calcule a probabilidade *a posteriori* desta partição auxiliar
fim repita
identifique a partição auxiliar mais provável
atualize a partição
fim algoritmo

Quadro 1: O Algoritmo

As Probabilidades de Transição entre as Partições Auxiliares

No problema em questão, em cada movimento (ciclo) do algoritmo, considerando as partições auxiliares (partições candidatas à nova partição), cada região A_i pode continuar na componente na qual está, pode migrar para alguma das outras componentes existentes ou pode formar uma nova componente. Para decidir qual das possibilidades é a mais provável se compara as razões $Q(t)$, definidas adiante pela Eq. (1.1). Estas razões envolvem as probabilidades de transição definidas como segue.

Suponha que estamos avaliando o que ocorre com a região A_i a qual, na partição atual $\rho' = \{S_1, ..., S_k\}$, pertence à componente S_j (é necessário identificar esta componente). Criam-se partições auxiliares ρ^* através da inclusão do índice j, que identifica o *cluster* ao qual A_i pertence, em cada componente existente em ρ', além de criar uma nova componente contendo apenas este elemento se for o caso. A razão ligada à cada partição auxiliar será:

$$Q(t) = \frac{P\Big(\rho^* = \{S_1^*, \ldots, \{S_t^*, i\}, \ldots, S_k^*\}\Big)}{P\Big(\rho^* = \{S_1^*, \ldots, S_b^*, S_{b+1}^*\}\Big)},\ t = 1, \ldots, b \text{ e } Q(b+1) = 1. \tag{1.1}$$

Na Eq. (1.1) use-se o fato de que

$$P\Big(\rho = \{S_1, ..., S_b\}\Big) \propto \prod_{k=1}^{b} L^{-1}(S_k)\Gamma(\alpha + \sum_{i \in S_k} x_i)\Big[\prod_{i \in S_k} x_i!\Big]^{-1}(n_{S_k} + \alpha)^{-(\alpha + \sum_{i \in S_k} x_i)},$$

em que $L(S_k)$ traz informação sobre o número de vizinhos das áreas pertencentes à componente S_k e α é um hiperparâmetro proveniente da distribuição *a priori* de $\boldsymbol{\theta}$.

1.3 Especificação e Estruturação Lógica do Programa

Sem perda de generalidade, neste trabalho será suposto um mapa com $n = 5$ regiões dispersas como na Figura 1.1, assumindo que os respectivos parâmetros são $\theta = (\theta_1 = 0.5, \theta_2 = 10, \theta_3 = 2, \theta_4 = 0.5, \theta_5 = 10)$. De acordo com a Figura 1.1, o número de vizinhos de cada região pode ser representado no vetor $\mathbf{L} = (2, 3, 4, 3, 2)$. Dada a discrepância entre estes θ_i, espera-se que o programa indique com alta probabilidade a partição formada por três *clusters* tais que $\rho = \{S_1 = (1, 4), S_2 = (2, 5), S_3 = (3)\}$. Para testar, uma amostra $\mathbf{X} = (X_1, ..., X_5)$ será gerada de distribuições Poisson com os parâmetros em θ.

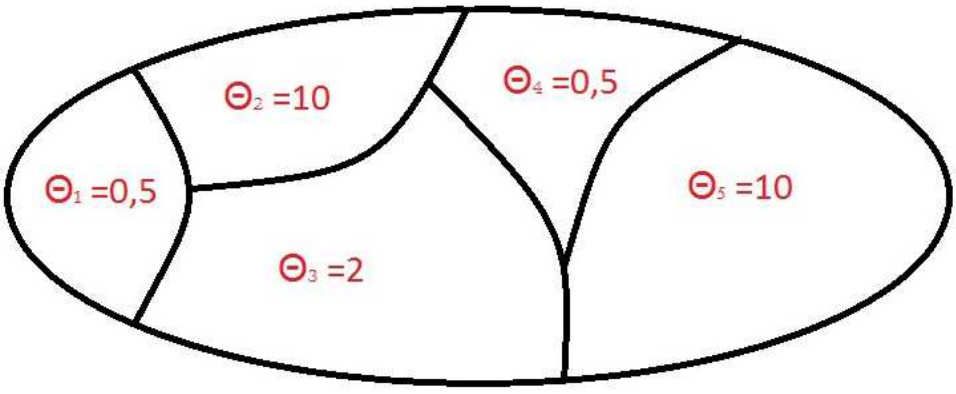

Figura 1.1: Regiões consideradas como exemplo

Além dos valores $\mathbf{X}$ e a estrutura de vizinhança $\mathbf{L}$, é necessário inicializar o parâmetro α referente à distribuição *a priori* para θ e uma partição inicial qualquer $\rho^{(0)} = \{S_1, \ldots, S_b\}$. Na atualização desta partição para algum $\rho^{(1)}$ se avalia o que acontece com cada região $A_i,\ i = 1, ..., 5$, no que chamamos de movimentos/ciclos (neste caso, são necessários 5 ciclos para uma atualização da partição). Em cada ciclo o elemento em questão é permutado entre

os *clusters* existentes além de ser considerado como um *cluster* unitário, formando o que chamamos de partições auxiliares. Calcula-se a probabilidade de cada possível partição auxiliar e escolhemos aquela mais provável baseado nas razões $Q(t)$, Eq. (1.1). Note que ao avaliarmos a possibilidade de se manter a mesma partição atual, temos $Q(t) = 1$. A função $\Gamma(.)$ envolvida foi calculada usando uma conhecida implementação em C++ (Press et al. 1986), obtida no site de disciplina (`http:http://www.est.ufmg.br/~fcruz/disciplinas/cce`).

Ao final dos ciclos se obtem uma nova partição e repetimos o processo quantas vezes forem desejadas. Tendo várias partições atualizadas podemos encontrar aquela mais provável e que deve indicar a correta separação das regiões mapa original em b *clusters*.

É trivial a escolha de uma estrutura de dados que facilite as trocas de índices ao longo dos ciclos (formação das partições auxiliares) e que ao mesmo tempo possibilite guardar informações sobre o número de *clusters* e o número de elementos dentro de cada *cluster*. Com este intuito, uma partição qualquer $\rho^{(t)} = \{S_1, ..., S_b\}$ será representada por um vetor de tamanho $n = 5$, chamado aqui de $\rho_.$, de forma que na posição i deste vetor esteja o índice referente ao *cluster* ao qual a região A_i pertence, ou seja, o vetor $\rho_.$ sempre terá n elementos que pertencem ao conjunto $1, ..., b$. Por exemplo, a partição $\rho = \{S_1 = (1, 4), S_2 = (2, 5), S_3 = (3)\}$, onde o número de *clusters* é $b = 3$, é representada por $\rho_. = \{1, 2, 3, 1, 2\}$. Além disto, considera-se uma representação auxiliar para $\rho_.$ que facilita a contagem do número de *clusters* e o número de elementos em cada um deles.

1.3.1 A Classe Proposta (*arquivo .hpp*)

Nesta Seção é apresentada a classe proposta. Sua implementação (extensão do *arquivo .hpp*) e o programa principal (*main*, *arquivo .cpp*) são apresentados nos Apêndices A e B deste relatório, respectivamente.

```
1  // file partEsp.hpp
2  #ifndef PART_ESP_HPP
3  #define PART_ESP_HPP
4  #include <stdio.h>
5  #include <math.h>
6  #include "gamma.cpp"
7
8  float alpha=2.0;
9
10 class ParticaoEsp {
11    int reg, nClusters, nContagens;
12    int  contagens,  vizinhos;
13    float razao, razaoMax;
14    int  IdxCluster;
15    int  IdxClusterMax;
16    int   regCluster;
17    int  nElemCluster;
18 //
19    int InicializePart(void);
```

```
20     int ImprimePrincip(void);
21     int ImprimeAuxiliar(void);
22     int GeraParticao(void);
23     int AtualizaAuxiliar(void);
24     int CalculaRazao(int orig, int dest);
25 public:
26     ParticaoEsp(void);                              //construtor
27     ~ParticaoEsp(void);                             //destrutor
28     int LeiaDados(char  nomeArqDados);//lê as o n  de regiões, as contagens e
         vizinhos
29     int ProcessaDados(void);              //aciona a geração das partições
30     int EscrevaResult(void);                 //imprime particao gerada
31 };
32 \\...
```

1.4 Resultados e Conclusões

A implementação da classe proposta na seção 1.3 é apresentada no Apêndice A e não está completamente concluída. Portanto, os resultados apresentados aqui são parciais. Por enquanto, tal implementação só funciona bem para o primeiro ciclo da geração de uma nova partição. Para os ciclos seguintes ainda há problemas quanto à correta especificação da estrutura auxiliar e, consequentemente, no cálculo das razões da Eq. (1.1), que definem a partição auxiliar mais provável. À propósito, pretende-se dar continuidade a este projeto, concluindo a implementação da classe proposta, corrigindo a estrutura e cálculos para os demais ciclos.

Para o exemplo considerado na seção 1.3, foi utilizado o seguinte vetor de contagens, $X = \{0, 10, 2, 0, 10\}$, sendo que o vetor de parâmetros verdadeiros é $\theta = (\theta_1 = 0.5, \theta_2 = 10, \theta_3 = 2, \theta_4 = 0.5, \theta_5 = 10)$ e o número de vizinhos de cada uma das $n = 5$ regiões é representado pelo vetor $\mathbf{L} = (2, 3, 4, 3, 2)$ como visto no mapa da Figura 1.1. Como partição inicial considerou-se $\rho^{(0)} = \{S_1 = (1), S_2 = (2), S_3 = (3), S_4 = (4), S_5 = (5)\}$ e, portanto, o vetor de representação alternativa é $\rho_{\cdot} = \{1, 2, 3, 4, 5\}$.

Neste caso, *no primeiro ciclo*, as partições auxiliares ρ^* possíveis, seus vetores de representação alternativa $\rho_{\cdot}$, o número b de *clusters* nestas partições e seus respectivos valores para $Q(t)$ calculados são apresentados na Tabela 1.1.

Tabela 1.1: Partições auxiliares e suas razões $Q(t)$ no primeiro ciclo

t	$\rho^{(*)}$	$\rho_{(.)}$	b	$Q(t)$
1	$\{S_1 = (1), S_2 = (2), S_3 = (3), S_4 = (4), S_5 = (5)\}$	(1,2,3,4,5)	5	1.0
2	$\{S_1 = (1,2), S_2 = (3), S_3 = (4), S_4 = (5)\}$	(1,1,2,3,4)	4	1.8769
3	$\{S_1 = (1,3), S_2 = (2), S_3 = (4), S_4 = (5)\}$	(1,2,1,3,4)	4	22.3342
4	$\{S_1 = (1,4), S_2 = (2), S_3 = (3), S_4 = (5)\}$	(1,2,3,1,4)	4	36.0015
5	$\{S_1 = (1,5), S_2 = (2), S_3 = (3), S_4 = (4)\}$	(1,2,3,4,1)	4	1.5415

Da Tabela 1.1 vemos que o algoritmo está selecionando as partições e calculando as razões conforme esperado, pois as regiões 1 (que é aquela avaliada no primeiro ciclo) e 4 foram unidas em um único *cluster*. Isto é o esperado porque as duas regiões são similares, no parâmetro e na contagem. Ambas possuem os menores valores para o parâmetro e para a contagem: $\theta = 0.5$ e $X = 0$. As regiões 2 e 5 são as mais diferentes da região 1 e, por isto, as razões relacionadas a estas regiões são baixas e de fato não é esperado que algumas destas regiões formem um *cluster* com a região 1. No entanto, não seria surpreendente se o algoritmo colocasse a região 1 em um *cluster* contendo a região 3, pois a diferença entre o parâmetro e a contagem observada para estas regiões não é muito discrepante: $\theta = 2$ e $X = 2$ para a região 3. De fato, a razão relativa à esta possibilidade (unir as regiões 1 e 3) é consideravelmente alta.

No geral, os resultados foram satisfatórios para o objetivo proposto e vemos que o algoritmo proposto por Crowley (1997) funciona bem na seleção das partições a serem consideradas no contexto do modelo partição produto.

Referências Bibliográficas

Bailey, T. C., Carvalho, M., Lapa, T., Souza, W. & Brewer, M. (2005), 'Modeling of under-detection of cases in disease surveillance.', *Annals of Epidemiology* **15**(5), 335–343.

Barry, D. & Hartigan, J. (1992), 'Product partition models for change point problems', *Journal of the American Statistical Association* **88**(421), 309–319.

Buzzy-Ferraris, G. (1993), *Scientific C++: Building Numerical Libraries the Object-Oriented Way*, Addison-Wesley Publishing Company Inc.

Crowley, M. E. (1997), 'Product partition models for normal means', *Journal of de American Statistical Association* **92**(437), 192–198.

Hartigan, J. A. (1990), 'Partition models', *Communication in Statistics-Theory & Methods* **19**(8), 2745–2756.

Oliveira, G. & Loschi, R. (2013), Mapeamento da mortalidade neonatal precoce em minas gerais: O uso da censura para contornar o problema do sub-registro. Monografia de conclusão do curso de Graduação em Estatística, UFMG.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, T. (1986), *Numerical Recipes*, Cambridge University Press.

A. Implementações (extensões do *arquivo .hpp*)

```
1  ...
2  ParticaoEsp::ParticaoEsp(void) {
3      fprintf(stdout, "Object ParticaoEsp created\n");
4  }
5  ParticaoEsp::~ParticaoEsp(void) {
6      fprintf(stdout, "Object ParticaoEsp destroied\n");
7      delete[] contagens;
8  }
9  int ParticaoEsp::ImprimePrincip(void) {
10   fprintf(stdout, "ParticaoEsp::ImprimePrincip()\n");
11   int i, j;
12    fprintf(stdout, "IdxCluster=[");
13    for (i=0; i<nContagens; i++) {
14       fprintf(stdout, "%d", IdxCluster[i]+1);
15       if(i<(nContagens-1)){
16          fprintf(stdout, ",");
17       }
18    }
19    fprintf(stdout, "]\n");
20    return 1;
21 }
22 int ParticaoEsp::InicializePart(void){
23   fprintf(stdout, "ParticaoEsp::InicializePart()\n");
24    IdxCluster = new int[nContagens];
25    IdxClusterMax = new int[nContagens];
26    regCluster = new int [nContagens];
27    nElemCluster = new int[nContagens];
28    int i;
29    for (i=0; i<nContagens; i++){
30       IdxCluster[i] = i;
31       regCluster[i] = new int[nContagens];
32       regCluster[i][0] = i;
33       nElemCluster[i] = 1;
34    }
35    nClusters = nContagens;
36    ImprimePrincip();
37    ImprimeAuxiliar();
38    razaoMax=1.0;
39    return 1;
40 }
41 int ParticaoEsp::ImprimeAuxiliar(void) {
42   fprintf(stdout, "ParticaoEsp::ImprimeAuxiliar()\n");
43   int i, j;
44    for (i=0; i<nContagens; i++) {
45       fprintf(stdout, "%d ", i+1);
46       for (j=0; j<nElemCluster[i];j++) {
47          fprintf(stdout, "%d ", regCluster[i][j]+1);
48       }
49       fprintf(stdout, "%d\n", nElemCluster[i]);
```

```
50     }
51     fprintf(stdout, "%d\n", nClusters);
52     return 1;
53 }
54 int ParticaoEsp::LeiaDados(char  nomeArqDados) {
55     const int LLENGTH = 256;
56   char Line[LLENGTH];
57     int i;
58   FILE  arqDados;
59     fprintf(stdout, "ParticaoEsp::LeiaDados\n");
60 //    open data file
61   fprintf(stdout, "Arquivo dados %s\n", nomeArqDados);
62   arqDados = fopen(nomeArqDados, "r");
63 //     read data file
64   fgets(Line, LLENGTH, arqDados);
65   fscanf(arqDados, "%d\n", &nContagens);
66   fprintf(stdout, "%d\n", nContagens);
67   fgets(Line, LLENGTH, arqDados);
68     contagens = new int[nContagens];
69     vizinhos = new int[nContagens];
70     for (i=0; i<nContagens; i++) {
71        fscanf(arqDados, "%d %d\n", &contagens[i], &vizinhos[i]);
72        fprintf(stdout, "%d %d\n", contagens[i], vizinhos[i]);
73     }
74   fclose(arqDados);
75   InicializePart();
76     ImprimeAuxiliar();
77   return 1;
78 }
79 int ParticaoEsp::GeraParticao(void){
80   fprintf(stdout, "ParticaoEsp::GeraParticao()\n");
81     for (int i=0; i<1; i++){       // só for(i<1) para fazer apenos o movimento
       para regiao 1
82        fprintf(stdout, "Inicio do Ciclo=%d", i+1);
83        for (int j=0; j<nContagens; j++){
84           if (i!=j){
85              IdxCluster[i] = IdxCluster[j];
86              AtualizaAuxiliar();
87              ImprimePrincip();
88              ImprimeAuxiliar();
89              CalculaRazao(i,j);
90              if (razao>razaoMax){
91                 razaoMax=razao;
92                 for (int k=0; k<nContagens; k++){
93                    IdxClusterMax[k] = IdxCluster[k];
94                 }
95              }
96           }
97        }
98        fprintf(stdout, "Resultado do Ciclo=%d:\tIdxClusterMax=[", i+1);
99        for (int k=0; k<nContagens; k++){
```

```
100             fprintf(stdout, "%d", IdxClusterMax[k]+1);
101             if(k<(nContagens-1)){
102                 fprintf(stdout,",");
103             }
104         }
105         fprintf(stdout, "], razaoMax=%f\n", razaoMax);
106         for (int k=0; k<nContagens; k++){
107             IdxCluster[k]=IdxClusterMax[k];
108         }
109         AtualizaAuxiliar();
110     }
111     return 1;
112 }
113 int ParticaoEsp::AtualizaAuxiliar(void){
114   fprintf(stdout, "ParticaoEsp::AtualizaAuxiliar()\n");
115   int pos;
116     nClusters = 0;
117     for (int i=0; i<nContagens; i++){
118         nElemCluster[i] = 0;
119     }
120     for (int i=0; i<nContagens; i++){
121         pos = nElemCluster[IdxCluster[i]];
122         regCluster[IdxCluster[i]][pos] = i;
123         nElemCluster[IdxCluster[i]]++;
124     }
125     for (int i=0; i<nContagens; i++){
126         if (nElemCluster[i] != 0){
127             nClusters++;
128         }
129     }
130     return 1;
131 }
132 int ParticaoEsp::CalculaRazao(int orig, int dest){
133   fprintf(stdout, "ParticaoEsp::CalculaRazao()\n");
134   int i, j;
135     float L1, L1Den, somat, somatDen;
136     L1=0; L1Den=0; somat=0; somatDen=0;
137     for (i=0; i<nElemCluster[dest]; i++){
138         L1+=vizinhos[regCluster[dest][i]];
139         somat+=contagens[regCluster[dest][i]];
140         if (i!=orig){
141             L1Den+=vizinhos[regCluster[dest][i]];
142             somatDen+=contagens[regCluster[dest][i]];
143         }
144     }
145     L1=1/L1;
146     L1Den=1/L1Den;
147     razao=(L1 Gamma(alpha+somat) pow((alpha+nElemCluster[dest]),-(alpha+somat))
          )/
148             (((1./vizinhos[orig]) Gamma(alpha+contagens[orig]) pow((alpha+1),-(
                  alpha+contagens[orig]))))
```

```
149           (L1Den Gamma(alpha+somatDen) pow((alpha+nElemCluster[dest]-1),-(alpha
                  +somatDen))));
150   fprintf(stdout, "razao=%f\n", razao);
151    return 1;
152 }
153 int ParticaoEsp::ProcessaDados(void){
154   fprintf(stdout, "ParticaoEsp::ProcessaDados()\n");
155    GeraParticao();
156    return 1;
157 }
158 int ParticaoEsp::EscrevaResult(void){
159     fprintf(stdout, "ParticaoEsp::EscrevaResult()\n");
160     return 1;
161 }
162 // end of file partEsp.hpp
163 #endif
```

B. Programa principal (*main*, *arquivo .cpp*)

```
1  // file partEsp.cpp
2  #include <stdio.h>
3  #include <math.h>
4  #include "partEsp.hpp"
5
6  int main(int argc, char   argv) {
7  // checar chamada ao programa
8     if (argc < 2) {
9        fprintf(stderr, "Uso: %s <entrada>\n", argv[0]);
10       return 0;
11    }
12    int aux = 0;
13    while ((argv[1][aux]!='\n')&&(argv[1][aux]!='\0')) aux++;
14    argv[1][aux] = '\0';
15    FILE  inputFile = fopen(argv[1],"r");
16    if (inputFile == NULL) {
17       fprintf(stderr, "%s: No such file\n", argv[1]);
18       return 0;
19    }
20    fclose(inputFile);
21 // entrar dados (ou simular)
22    ParticaoEsp minhaPart;
23    minhaPart.LeiaDados(argv[1]);
24    minhaPart.ProcessaDados();
25    minhaPart.EscrevaResult();
26    return 1;
27 }
28 // end of file partEsp.cpp
```

Método de Reamostragem *Bootstrap* via C++

José Carlos da Rocha
Departamento de Estatística - ICEx - UFMG
31270-901 - Belo Horizonte - MG
E-mail: carlosmarcell2013@gmail.com

Resumo

Em inferência estatística, deseja-se quantificar o erro cometido ao se estimar um parâmetro de interesse θ através de $\hat{\theta}$. Uma estratégia usual para a busca de medidas de incerteza, que expressem este erro, é a estimação do intervalo de confiança de $\hat{\theta}$. Entretanto, métodos analíticos para a obtenção destas medidas nem sempre são disponíveis, ou constituem processos altamente complexos. Enquanto isso, métodos assintóticos, nos quais a construção de intervalos de confiança é baseada, dependem de aproximações nem sempre alcançadas. Neste contexto, o método *bootstrap* constitui uma eficiente alternativa, fornecendo estimativas do erro padrão de $\hat{\theta}$ livres de complexidades algébricas e possibilitando a obtenção inferências sem a necessidade de pressupostos sobre a distribuição do estimador. Neste trabalho, apresentamos de forma concisa a obtenção do intervalo de confiança de $\hat{\theta}$ através do método *bootstrap*, utilizando o *software* C++ a fim de fazer uma comparação entre o desempenho do R e do C++.

2.1 Introdução

A ideia básica da técnica *bootstrap* de reamostragem é tratar a amostra original como se fosse a população de interesse e, retirar várias amostras com reposição da mesma, ou seja, reamostrar a amostra original com reposição e, para cada reamostra, calcular a estimativa de interesse, estimar características populacionais e fazer inferência sobre a população tais como intervalos de confiança, erro padrão e *etc*. Além disso, a distribuição da população finita representada pela amostra pode ser encarada como uma pseudo-população, com características análogas às da verdadeira população.

O método foi introduzido por Efron (1979). Os métodos de *bootstrap* são uma classe de métodos de Monte Carlo não-paramétricos que estimam a distribuição da população

por reamostragem. O termo "*bootstrap*" pode ser dirigido a *bootstrap* não-paramétrico ou *bootstrap* paramétrico, com a distinção de que no primeiro, a distribuição da população de onde se originou a amostra não é conhecida, enquanto que se tem conhecimento da distribuição dos dados no segundo caso. É importante notar que o *bootstrap* gera amostras aleatoriamente a partir da distribuição empírica da amostra.

2.1.1 Objetivos

O objetivo deste trabalho é apresentar a obtenção de intervalos de confiança percentílico de $\hat{\theta}$ através do método *bootstrap* e verificar a cobertura do seu valor teórico, utilizando o *software* C++ e fazer uma comparação entre o desempenho do R e do C++.

2.1.2 Organização

Este texto está organizado da seguinte forma. Na seção 2.2, apresentamos a metodologia. Resultados experimentais são apresentados na seção 2.3. Conclusões finais encerram este relatório na seção 2.4, com sugestões de trabalhos futuros na área.

2.2 Metodologia

2.2.1 O Estimador *Bootstrap* do Erro Padrão

Seja uma amostra aleatória baseada em n observações independentes $x_1, x_2, ..., x_n$. O erro padrão de uma média $\overline{x}$ baseada nesta amostra é estimado pela seguinte expressão:

$$\widehat{ep}(\overline{x}) = \sqrt{\frac{s^2}{n}}, \tag{2.1}$$

em que:

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1} \tag{2.2}$$

é o estimador não viciado da variância. Nota-se que o erro padrão não é uma estimativa de uma quantidade pertinente a uma população, mas uma medida da incerteza da média amostral vista como uma estimativa da média populacional (Altman 1991). A expressão do erro padrão, Eq. (2.1), deixa claro que a magnitude desta incerteza diminui conforme o tamanho da amostra, n, aumenta.

Observada uma amostra aleatória de tamanho n, oriunda de uma distribuição F, define-se uma função distribuição empírica $\widehat{F_n}$ como uma distribuição discreta, que atribui probabilidade $1/n$ a cada valor $i = 1, \ldots, n$. Uma amostra *bootstrap* $X^{(*)} = x_1^*, ..., x_n^*$ é obtida reamostrando-se aleatoriamente B vezes, com reposição, as observações $x = (x_1, \ldots, x_n)$ das

quais será construída, a partir das B amostras *bootstrap* $x_1^*, ..., x_n^*$, a função $\widehat{F}_n$, que para um t fixo, pode-se escrever $\widehat{F}_n$ na forma:

$$F_n^*(t) = \frac{1}{n}\sum_{i=1}^{n} 1_{\{\hat{\theta}^{(b)} \leq t\}}, \tag{2.3}$$

em que $F_n^*(t) = \widehat{F}_n$ é a função distribuição empírica das réplicas $\hat{\theta}^{(1)}, \cdots, \hat{\theta}^{(b)}$.

Pesquisas sobre o assunto indicam que quando o número de reamostras é muito grande, $\widehat{F}_n$ converge uniformemente para F, o que justifica a Eq. (2.3), para um t fixo com esperança e variância

$$E(\widehat{F}_n(t)) = n^{-1}\sum_{j=1}^{n} E1_{\{x_j \leq t\}} =^{iid} P(x_1 \leq t) = F(t)$$

e

$$V(\widehat{F}_n(t)) = \frac{F(t) - F(t)^2}{n},$$

respectivamente.

Assim, mostra-se que para um t fixo, $\widehat{F}_n(t)$ é um estimador não viciado de $F(t)$ e, a medida que n cresce, a variância de $\widehat{F}_n(t)$ decresce.

2.2.2 O Estimador do Intervalo de Confiança Percentílico *Bootstrap* para a Média

O intervalo de confiança de nível $(1-\alpha)100\%$ é estimado a partir dos quantis da distribuição empírica estimada na seção anterior, ordenando o vetor de todos os parâmetros estimados na reamostra e calculando os percentis $G^{*-1}(\alpha/2)$ e $G^{*-1}(1-\alpha/2)$, em que G^* é a acumulada da função distribuição empírica $\widehat{F}_n$ das réplicas $\hat{\theta}^{(1)}, \cdots, \hat{\theta}^{(b)}$.

De forma resumida seguem os passos para fazermos uma reamostragem *bootstrap* a fim de estimarmos o intervalo de confiança percentílico de nível 1-$\propto$ para o estimador da média:

- Selecione B amostras *bootstrap* de forma independente, $X^{*b} = X^{*1}, X^{*2}, \ldots, X^{*B}$;

- Estima-se θ em cada uma destas amostras através de $\widehat{\theta}^{*b} = S(X^{*b}),\ b = 1, 2, \ldots, B$;

- Ordene os e calcule os limites inferiores e superiores através da distribuição empírica;

- O método verificará a cobertura do valor teórico a um nível $(1-\alpha)100\%$.

Por fim, baseado nos passos acima, tem-se a classe apresentada na Figura 2.1, a ser implementada no software C++. A Figura 2.2 apresenta o arquivo com o programa principal (*main*).

```
69  class Boot {
70  private:
71      int tamanho, B, semente;
72      float *amostra;
73      float *reamostra;
74      float *thetaBoot;
75      void Reamostre(void);
76      void Ordena(void);
77      float Theta(void);
78      float Percent(float p);
79  public:
80      Boot(void);                          // construtor default
81      ~Boot(void);                         // destrutor
82      int LeiaDados(char *entra);          // leitura dos dados
83      void IC(float nivel, float *Linf, float *Lsup); // calculo do IC
84  };
```

Figura 2.1: Classe para implementação em C++ do *bootstrap*

```
4   int main(int argc, char** argv) {
5   //  checar chamada ao programa
6       if (argc < 2) {
7           fprintf(stderr, "Uso: %s <arq_entrada.ext>\n", argv[0]);
8           return 0;
9       }
10      int aux = 0;
11      while ((argv[1][aux]!='\n')&&(argv[1][aux]!='\0')) aux++;
12      argv[1][aux] = '\0';
13      FILE *inputFile = fopen(argv[1],"r");
14      if (inputFile == NULL) {
15          fprintf(stderr, "%s: No such file\n", argv[1]);
16          fclose(inputFile);
17          return 0;
18      }
19      Boot umBoot;
20  //  leia entrada
21      umBoot.LeiaDados(argv[1]);
22  //  calcule estimativas bootstrap e verifique a cobertura
23      float nivel=0.90;
24      float Linf, Lsup;
25      float thetaTeor=0.0;
26      int cob=0;
27      int mcarlo=1000;
28      for (int i=0; i<mcarlo; i++) {
29          umBoot.IC(nivel, &Linf, &Lsup);
30          if ((thetaTeor>=Linf)&&(thetaTeor<=Lsup)) cob++;
31      }
32  //  escreva resultados
33      fprintf(stdout, "Cobertura (nivel %3.2f em %d replicacoes)=%d\n", nivel, mcarlo, cob);
34      fprintf(stdout, "Ultimo IC=[%f,%f]\n", Linf, Lsup);
35      return 1;
36  }
```

Figura 2.2: Classe para implementação em C++ do *bootstrap*

2.3 Resultados Experimentais

Os resultados obtidos são apresentados a seguir. Utilizando-se o R, duas amostras foram geradas, provenientes de uma distribuição exponencial, com valor esperado igual a 1/3. Foram calculados 1000 intervalos de 90% de confiança ($1-\alpha = 0,90$). As coberturas obtidas são apresentadas a seguir.

```
Object Boot created
Boot::LeiaDados
Arquivo de entrada: expo320.txt
Cobertura (nivel 0.90 em 1000 replicacoes)=1000
Ultimo IC=[0.204509,0.433365]
Object Boot destroied

Object Boot created
Boot::LeiaDados
Arquivo de entrada: expo3100.txt
Cobertura (nivel 0.90 em 1000 replicacoes)=1000
Ultimo IC=[0.271751,0.371119]
Object Boot destroied
```

2.4 Conclusões e Observações Finais

Dos resultados apresentados, conclui-se que a cobertura do intervalo de confiança *bootstrap* foi de 100%, e não do esperado, que era de 90%. Melhores implementações do *bootstrap* precisam ser analisadas para que possamos ter resultados plausíveis quanto à cobertura desses intervalos.

Referências Bibliográficas

Altman, D. G. (1991), *Practical statistics for medical research*, Chapman and Hall, London.

Efron, B. (1979), 'Bootstrap methods: Another look at the jackknife', *The Annals of Statistics* **7**, 1–26.

Efron, B. & Tibshirani, R. (1994), *An Introduction to the Bootstrap*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, London.

Modelos Dinâmicos Lineares Generalizados: Implementação para a Distribuição de Poisson

Jéssica da Assunção Almeida
Departamento de Estatística - ICEx - UFMG
31270-901 - Belo Horizonte - MG
E-mail: jessica@est.grad.ufmg.br

Resumo

Nesta etapa do projeto, além de ser definido detalhadamente a metodologia de modelos lineares generalizados dinâmicos, especificadamente para a distribuição de Poisson, a implementação em C++, desenvolvida ao longo da disciplina de Computação Científica e Estatística I, também é apresentada. Ainda, alguns resultados, obtidos através da implementação são analisados.

3.1 Introdução

Uma série temporal é uma série de observações tomadas sequencialmente no tempo. A modelagem deste tipo de dados é abordada, especialmente, nos modelos dinâmicos, cuja natureza demanda que a incerteza devido à passagem do tempo seja avaliada.

Seja Y_t um vetor $(n \times 1)$ referente as observações de uma série temporal nos tempos $t = 1, 2, ..., n$. A subclasse mais popular entre os modelos dinâmicos é conhecida como modelo linear dinâmico normal (DLM) e pode ser definido como:

$$Y_t = F_t' \, \theta_t + v_t \ , \qquad v_t \sim N[0, V_t]$$
$$\theta_t = G_t \, \theta_{t-1} + w_t \ , \qquad w_t \sim N[0, W_t]$$

em que F_t' é a matriz das p variáveis independentes $(p \times n)$, θ_t é o vetor de regressão, v_t é o erro observacional, G_t é a matriz $(n \times n)$ de evolução e w_t é o erro de evolução.

Entretanto, em uma gama de situações não é razoável a suposição de normalidade. Modelos lineares dinâmicos podem ser generalizados retirando a condição de normalidade. Esse tipo de generalização é útil ao modelarmos, por exemplo, uma série temporal discreta, em

que Y_t represente o número de sucessos em um determinado número de realizações de um experimento, ou, em que Y_t represente contagens.

Neste trabalho temos como objetivo implementar a metodologia de modelos dinâmicos lineares generalizados em linguagem *C++* para a distribuição de Poisson, a fim de obtermos algum ganho, principalmente, relacionado ao tempo, quando comparado a softwares estatísticos, como o *R*.

O trabalho está organizado da seguinte forma. Na seção 3.2 o modelo linear generalizado dinâmico é definido formalmente, os passos necessários para a atualização do modelo são desenvolvidos e o caso específico Poisson é mostrado. Na seção 3.3 os passos da implementação são apresentados, ou seja, o algoritmo proposto e a classe proposta são mostrados. Por fim, os resultados obtidos são analisados na seção 3.4.

3.2 Modelos Lineares Generalizados Dinâmicos

Considere observações escalares, contínuas ou discretas, de uma série temporal $Y_t, t = 1, 2, \ldots, n$, pertencentes à uma distribuição de probabilidade, membro da família exponencial. Ou seja,

$$P(Y_t|\eta_t, V_t) \;=\; \exp\Big\{V_t^{-1}\big[y_t(Y_t)\eta_t \;-\; a(\eta_t)\big]\Big\}b(Y_t, V_t), \tag{3.1}$$

em que η_t é o parâmetro canônico, $V_t > 0$ é um parâmetro de escala, sendo que o parâmetro de precisão é definido como $\phi_t = V_t^{-1}$, e $y_t(.)$, $a(.)$ e $b(.,.)$ são funções conhecidas.

De acordo com Dobson (2002), a função $a(\eta_t)$ é duas vezes diferenciável, o que implica em

$$\mu_t \;=\; E(y_t(Y_t)|\eta_t, V_t) \;=\; \frac{da(\eta_t)}{d\eta_t} \;=\; \dot{a}(\eta_t),$$

e

$$\text{Var}\left[y_t(Y_t)|\eta_t, V_t\right] \;=\; V_t\ddot{a}(\eta_t).$$

Os modelos lineares generalizados (Nelder & Wedderburn 1972) consideram os dados condicionalmente independentes com distribuição comum, Eq. (3.1). Essa classe de modelos tem como ideal relacionar os parâmetros canônicos à uma função linear das variáveis explicativas. No contexto de séries temporais, o uso de modelos lineares generalizados é apropriado aplicando os modelos lineares generalizados dinâmicos, definidos a seguir.

3.2.1 Definição

Defina as seguintes quantidades no tempo t:

- θ_t, um vetor de estados no tempo t n -dimensional;

- F_t, um vetor conhecido de regressão n-dimensional;

- G_t, uma matriz conhecida de evolução $n \times n$;

- ω_t, um vetor de erros de evolução, n-dimensional, possuindo média zero e matriz de variância conhecida W_t, denotado por $\omega_t \sim [0, W_t]$;

- $\lambda = F_t'\theta_t$, uma função linear dos parâmetros do vetor de estados;

- $g(\eta_t)$, uma função conhecida, monótona e contínua, que liga η_t aos reais.

Assim, o modelo linear generalizado dinâmico (DGLM) para a série temporal Y_t é definido pelos seguintes componentes:

- Modelo observacional,

$$P(Y_t|\eta_t) \quad e \quad g(\eta_t) \;=\; \lambda_t \;=\; F_t'\theta_t, \tag{3.2}$$

- Equação de evolução,

$$\theta_t = G_t\theta_{t-1} + \omega_t \quad com \quad \omega_t \sim [0, W_t]. \tag{3.3}$$

3.2.2 Atualização

Os componentes chave da análise proposta por West et al. (1985) são desenvolvidos a seguir. A definição do modelo é completada supondo ω_t especificada apenas em termos de média e variância. Além disso, suponha

$$(\theta_{t-1}|D_{t-1}) \;\sim\; [m_{t-1}, C_{t-1}], \tag{3.4}$$

ou seja, a forma da distribuição *a posteriori* de θ_t também não é especificada, apenas os momentos.

Segue da Eq. (3.3) que os momentos *a priori* para θ_t são

$$(\theta_t|D_{t-1}) \;\sim\; (a_t, R_t), \tag{3.5}$$

em que $a_t = G_t m_{t-1}$ e $R_t = G_t C_{t-1} G_t' + W_t$.

O preço a pagar pela generalização, ou seja, por não assumir normalidade é a perda de informação. Tendo então, as devidas especificações em termos dos momentos, a estimação bayesiana é feita da forma a seguir.

- **Passo 1 - Distribuição *a priori* para λ_t:**

  De (3.5), temos

$$\begin{pmatrix} \lambda_t \\ \theta_t \end{pmatrix} \Bigg| \, D_{t-1} \;\sim\; \left[\begin{pmatrix} f_t \\ a_t \end{pmatrix} \;,\; \begin{pmatrix} q_t & F_t'R_t \\ R_tF_t & R_t \end{pmatrix} \right], \tag{3.6}$$

  em que

$$f_t = F_t'a_t \quad \text{e} \quad \mathrm{q_t} = \mathrm{F_t'R_tF_t}.$$

- **Passo 2 - Previsão um passo a frente:**

A informação amostral relevante para prever Y_t é completamente resumida na distribuição marginal *a priori* para $(\eta_t|D_{t-1})$,

$$p(Y_t|D_{t-1}) \;=\; \int p(Y_t|\eta_t)p(\eta_t|D_{t-1})d\eta_t. \tag{3.7}$$

Assumimos então a forma conjugada para a distribuição *a priori* de η_t, isto é,

$$p(\eta_t|D_{t-1}) \;=\; c(r_t, s_t)\exp\left[r_t\eta_t \;-\; s_t a(\eta_t)\right]. \tag{3.8}$$

Os parâmetros r_t e s_t são escolhidos de forma que sejam consistentes com os momentos para λ_t, já que $\lambda_t = g(\eta_t)$,

$$E[g(\eta_t)|D_{t-1}] = f_t \quad \text{e} \quad \mathrm{Var[g(\eta_t)|D_{t-1}]} = \mathrm{q_t}. \tag{3.9}$$

Logo, a previsão um passo a frente será dada, por

$$p(Y_t|D_{t-1}) \;=\; \frac{c(r_t, s_t)b(Y_t, V_t)}{c(r_t + \phi_t Y_t, s_t + \phi_t)}. \tag{3.10}$$

Passo 3 - Atualização de η_t:

A distribuição a posteriori para η_t é dada por

$$p(\eta_t|D_t) \;=\; c(r_t + \phi_t Y_t, s_t + \phi_t)\exp[(r_t + \phi_t Y_t)\eta_t \;-\; (s_t + \phi_t)a(\eta_t)]. \tag{3.11}$$

Por analogia temos

$$E[g(\eta_t)|D_t] = f_t^* \quad \text{e} \quad \mathrm{Var[g(\eta_t)|D_t]} = \mathrm{q_t^*}. \tag{3.12}$$

Passo 4 - Atualização de θ_t:

O objetivo final da atualização é calcular a distribuição *a posteriori* de θ_t. Com auxílio do *Linear Bayesian Estimation* (LBE), utilizado para a estimação apenas em termos dos momentos, temos o seguinte:

$$(\theta_t|D_t) \;\sim\; [m_t, C_t], \tag{3.13}$$

em que

$$m_t \;=\; a_t \;+\; R_t F_t(f_t^* - f_t)/q_t \tag{3.14}$$

e

$$C_t \;=\; R_t \;-\; R_t F_t F_t' R_t(1 - q_t^*/q_t)/q_t. \tag{3.15}$$

3.2.3 Distribuição de Poisson

Em modelos lineares generalizados a distribuição de Poisson é associada a dados de contagem. Suponha $Y_t \sim \text{Poisson}(\mu_t)$ uma série temporal associada à contagem. Temos que para $\mu_t > 0$, $Y_t|\mu_t$ tem função de probabilidade dada por

$$P(Y_t|\mu_t) \;=\; \frac{e^{-\mu_t}\mu_t^{Y_t}}{Y_t!} \;=\; \exp\{Y_t \log(\mu_t) \;-\; \mu_t\}\; \frac{1}{Y_t!}, \quad y_t = 0, 1, ..., \tag{3.16}$$

Note que este é um caso especial da Eq. (3.1) em que $V_t^{-1} = 1$, $y_t(Y_t) = Y_t$, $\eta_t = \log(\mu_t)$, $a(\eta_t) = e^{\eta_t}$ e $b(Y_t, V_t) = 1/Y_t!$.

Temos que as distribuições *a priori* conjugada e *a posteriori* para μ_t são dadas respectivamente por:

$$(\mu_t|D_{t-1}) \sim \text{Gama}(r_t, s_t), \tag{3.17}$$

$$(\mu_t|D_t) \sim \text{Gama}(r_t + Y_t, s_t + 1). \tag{3.18}$$

Daí

$$E(\mu_t|D_{t-1}) = \frac{r_t}{s_t} \quad \text{e} \quad \text{Var}(\mu_t|\text{D}_{\text{t}-1}) = \frac{\text{r}_\text{t}}{\text{s}_\text{t}^2},$$

$$E(\mu_t|D_t) = \frac{r_t + Y_t}{s_t + 1} \quad \text{e} \quad \text{Var}(\mu_t|\text{D}_\text{t}) = \frac{\text{r}_\text{t} + \text{Y}_\text{t}}{(\text{s}_\text{t} + 1)^2}.$$

Neste caso

$$\lambda_t = \eta_t = \log(\mu_t) = F_t'\theta_t.$$

Portanto, baseados em aproximações numéricas teremos

$$E(\eta_t|D_{t-1}) \;=\; f_t \;\approx\; \log\left(\frac{r_t}{s_t}\right) \quad \Rightarrow \quad s_t = \frac{\exp(-f_t)}{q_t},$$

$$E(\eta_t|D_{t-1}) \;=\; q_t \;\approx\; \frac{1}{r_t} \quad \Rightarrow \quad r_t = \frac{1}{q_t},$$

$$E(\eta_t|D_t) \;=\; f_t^* \;\approx\; \log\left(\frac{r_t + Y_t}{s_t + 1}\right),$$

$$E(\eta_t|D_t) \;=\; q_t^* \;\approx\; \frac{1}{r_t + Y_t}.$$

Com os valores de r_t, s_t, f_t^* e q_t^* aproximados podemos calcular os momentos de $\theta_t|D_t$ conforme a Eq. (3.13), assim como a previsão um passo a frente como na Eq. (3.10).

3.3 Implementação

A implementação do problema proposto foi feita em linguagem *C++* e é apresentada através de um algoritmo estruturado, seguida de sua programação e uma classe proposta para o problema nas seções seguintes.

3.3.1 Algoritmo

O algoritmo proposto para a metodologia de modelos lineares generalizados dinâmicos é apresentado na Figura 3.1. A esquematização é feita da seguinte forma: inicialmente são definidas as variáveis utilizadas no problema e o conjunto de dados lido, sendo que este conjunto é composto por um vetor de variáveis respostas e por uma matriz de variáveis explicativas. Estabelecidas as variáveis e os dados é feita a atualização para o tempo 1, já que neste caso não há equação de evolução, e em seguida a atualização é feita para os demais tempos. Por fim, os resultados de interesse, neste caso, o vetor de médias *a posteriori* de θ, são escritos.

A implementação do algoritmo em *C++* pode ser vista na Figura 3.2 através do arquivo principal do programa construído. Note que sua estruturação lógica é similar ao do algoritmo, ou seja, através da função *meuDGLM.LeiaDados()* a leitura dos dados é feita. Na função *meuDGLM.ProcesseDados()* são feitas as atualizações necessárias e por fim, os resultados são escritos através da função *meuDGLM.EscreveResults()*. A definição das variáveis é feita na classe proposta

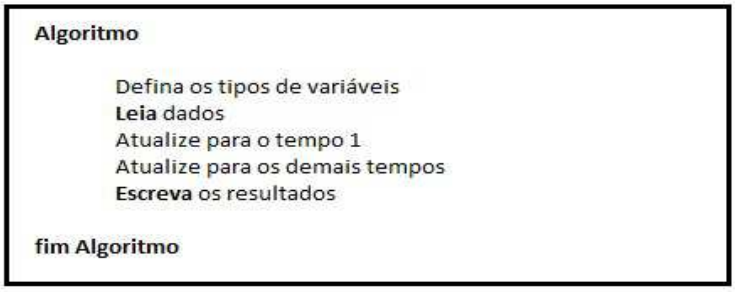

Figura 3.1: Algoritmo para a metodologia de modelos lineares generalizados dinâmicos.

3.3.2 Classe

A classe proposta para implementação do problema em C++ é apresentada na Figura 3.3. Tal classe consiste na definição das variáveis de entrada e de saída e na definição de métodos públicos, que possuem a função de ler os dados, processá-los e escrever os resultados de interesse. A implementação da classe proposta é apresentada no apêndice.

```
#include <stdio.h>
#include "DGLM.hpp"

int main(int argc, char** argv) {

    if (argc<3) {
        fprintf(stderr, "Uso: %s <entrada> <saida>\n", argv[0]);
        return 0;
    }
    int aux=0;
    while ((argv[1][aux]!='\n')&&(argv[1][aux]!='\0')) aux++;
    argv[1][aux] = '\0';
    aux=0;
    while ((argv[2][aux]!='\n')&&(argv[2][aux]!='\0')) aux++;
    argv[2][aux] = '\0';
    FILE *inputFile = fopen(argv[1],"r");
    if (inputFile == NULL) {
        fprintf(stderr, "%s: No such file\n", argv[1]);
        return 0;
    }
    fclose(inputFile);

    DGLM meuDglm;
    meuDglm.LeiaDados(argv[1]);

    meuDglm.ProcesseDados();

    meuDglm.EscrevaResults(argv[2]);
    return 1;
}
```

Figura 3.2: Arquivo principal do programa construído em *C++*.

```
#ifndef _DGLM_H
#define _DGLM_H

#include <math.h>
#include <stdio.h>

class DGLM {
    FILE *fileData;
    FILE *fileResult;
    // dados de entrada
    int tam;
    int ncov;
    float *Yt;
    float **Ft;          // Matriz de covariáveis
    // definição do modelo
    float delta;         // Fator de desconto
    float **G;           // Matriz de evolução
    float sigma2;        // Variância a priori de teta
    // Momentos a priori
    float **a;
    float ***R;
    // Momentos a posteriori
    float **m;
    float ***C;
public:
    DGLM(void);            // construtor default
    ~DGLM(void);           // destrutor
    int LeiaDados(char *nomeArqDados);
    int ProcesseDados(void);
    int EscrevaResults(char *nomeArqSaida);
};
```

Figura 3.3: Classe proposta em linguagem *C++*

3.4 Resultados

O objetivo final da análise de um modelo linear generalizado dinâmico é a obtenção dos momentos a posteriori. Com esses resultados pode-se fazer previsão k passos a frente, que é

um interesse comum ao tratar dados de séries temporais.

Neste trabalho, a implementação proposta na seção 3.3 foi feita para a distribuição de Poisson. Para isso simulamos, em R, uma série temporal de contagem de tamanho 100, ou seja, com distribuição de Poisson, assim como uma matriz de 3 variáveis explicativas composta por : uma variável correspondente ao intercepto, uma variável binária (gerada da distribuição binomial), que pode ser interpretada como um fator que separa grupos, e uma variável contínua (gerada de uma distribuição normal) que pode ser interpretada como alguma medição.

Ao gerarmos de um série temporal precisamos saber seu comportamento, ou seja, o vetor real de θ é conhecido. Portanto, utilizamos o vetor de médias *a posteriori*, calculado em *C++*, de θ como estimativa pontual e comparamos os valores reais e estimados na Figura 3.4.

É visto que os resultados são razoáveis e que o valor estimado, acompanha, com certa oscilação, o valor real. A implementação em *C++* se mostrou eficiente, entretanto a especificação da matriz W_t é complicada e para isto utilizamos um fator de desconto, que em base, representa a quantidade de informação que é deixada passar de um intervalo para o outro. Logo, os resultados da Figura 3.4 podem ser melhorados ou piorados de acordo com a especificação do fator de desconto. Aqui utilizamos um valor igual a 0,6.

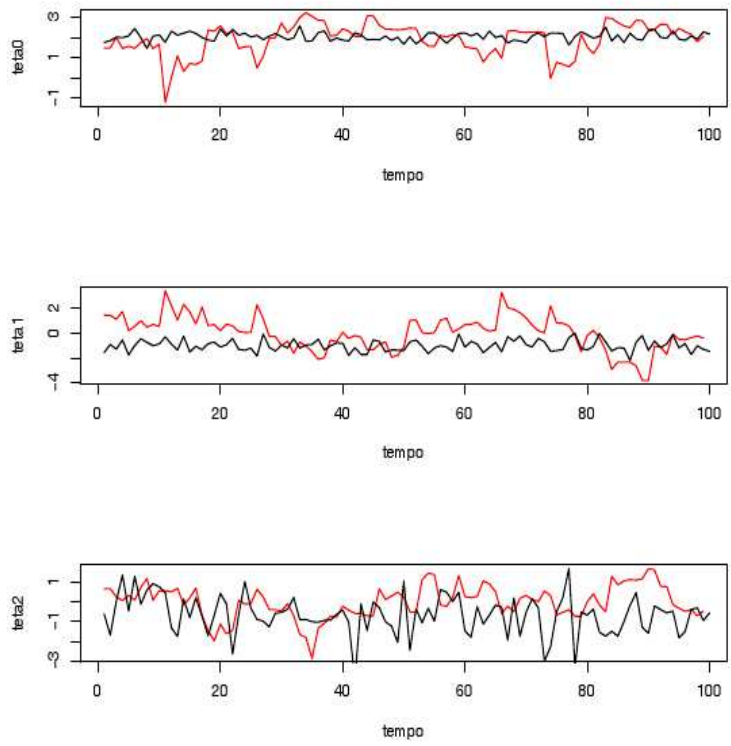

Figura 3.4: Valores reais de θ (em preto) e valores estimados de θ (em vermelho).

3.5 Considerações Finais

A implementação em *C++* se mostrou eficiente e os resultados de se comportaram dentro do esperado. Entretanto, o problema proposto lida com matrizes que devem ser manipuladas ao longo da atualização. Esse tipo de manipulação é extremamente custosa em *C++*, já que, elas devem ser implementadas pelo programador. Portanto, concluímos que apesar de termos obtidos resultados satisfatórios e ser conhecido o ganho de tempo, o uso dessa linguagem deve ser repensado em trabalhos futuros.

Referências Bibliográficas

Dobson, A. J. (2002), *An Introduction to Generalized Linear Models*, Chapman & Hall/CRC.

Nelder, A. & Wedderburn, R. W. M. (1972), 'Generalized linear models', *J. R. Statist. Soc. A* **135, Part 3**, 370.

West, M., Warisson, J. P. & Migon, H. S. (1985), 'Dynamic generalized linear models and bayesian forecasting', *Journal of the American Statistical Association* pp. 83 – 93.

A. Implementação da Classe Proposta

```
1
2 DGLM::DGLM(void) {
3     fprintf(stdout, "Object DGLM created\n");
4 }
5
6 DGLM::~DGLM(void) {
7     fprintf(stdout, "Object DGLM destroied\n");
8 }
9
10 int DGLM::LeiaDados(char  nomeArqDados) {
11    const int LLENGTH = 256;
12     char Line[LLENGTH];
13     int i, j;
14     FILE  arqDados;
15    fprintf(stdout, "DGLM::LeiaDados\n");
16 // abrir arquivo de dados
17    fprintf(stdout, "Arquivo de dados: \%s\n", nomeArqDados);
18    arqDados = fopen(nomeArqDados, "r");
19 // determinar numero de covariaveis e tamanho da amostra
20    fgets(Line, LLENGTH, arqDados);
21    ncov=0; i=0;
22    while ((Line[i]!='\n')&&(Line[i]!='\0')) {
23       if (Line[i]==' ') {
24          ncov++;
25       }
26       i++;
27    }
28 // fprintf(stdout, "ncov=\%d\n", ncov);
29    tam=0;
30    while (fgets(Line, LLENGTH, arqDados)) {
31       tam++;
32    };
33 // fprintf(stdout, "tam=\%d\n", tam);
34    Yt = new float[tam];
35     Ft = new float [ncov];
36     for (int i=0; i<ncov; i++) {
37       Ft[i]= new float[tam];
38     }
39    rewind(arqDados);
40     fgets(Line, LLENGTH, arqDados);
41     for (int i=0; i<tam; i++) {
42       fscanf(arqDados, "\%f", &Yt[i]);
43       fprintf(stdout, "\%f ", Yt[i]);
44       for (int j=0; j<ncov; j++) {
45          fscanf(arqDados, "\%f", &Ft[j][i]);
46          fprintf(stdout, "\%f ", Ft[j][i]);
47       }
48       fscanf(arqDados, "\n");
49       fprintf(stdout, "\n");
```

```
50      }
51      fclose(arqDados);
52      return 0;
53  }
54
55  int DGLM::ProcesseDados(void) {
56     fprintf(stdout, "DGLM::ProcesseDados\n");
57     // variaveis locais
58     int i, j, t, k;
59     float ftj0, qtj0, ftj1, qtj1, rtj, stj,  aux,   aux2,   aux3,   aux4;
60     // alocacao das variaveis
61     G = new float [ncov];
62      a = new float [ncov];
63      R = new float  [ncov];
64      m = new float [ncov];
65      C = new float  [ncov];
66     aux = new float[ncov];
67     aux2 = new float [ncov];
68     aux3 = new float [ncov];
69     aux4 = new float [ncov];
70      for (i=0; i<ncov; i++) {
71        G[i]= new float[ncov];
72        a[i]= new float[tam+1];
73        R[i]= new float [ncov];
74        m[i]= new float[tam];
75        C[i]= new float [ncov];
76        aux2[i]= new float[ncov];
77        aux3[i]= new float[ncov];
78        aux4[i]= new float[ncov];
79        for (j=0; j<ncov; j++) {
80           R[i][j]= new float[tam+1];
81           C[i][j]= new float[tam];
82        }
83     }
84     // inicializacoes
85     delta = 0.6;
86     sigma2 = 100;
87      for (i=0; i<ncov; i++) {
88        for (j=0; j<ncov; j++) {
89           G[i][j]=0.0;
90        }
91        G[i][i]=1.0;
92      }
93      t=0;
94      // fixando os momentos a priori
95      for (i=0; i<ncov; i++) {
96        a[i][t]=0.0;
97        for (j=0; j<ncov; j++) {
98           R[i][j][t]=0.0;
99        }
100       R[i][i][t]=sigma2;
```

```
101     }
102     for (t=0; t<tam; t++) {
103        // fixando os momentos a priori
104        ftj0=0.0;
105        qtj0=0.0;
106        for (i=0; i<ncov; i++) {
107           ftj0+=Ft[i][t] a[i][t];
108           aux[i]=0.0;
109           for (j=0; j<ncov; j++) {
110              aux[i]+=Ft[j][t] R[j][i][t];
111           }
112           qtj0+=aux[i] Ft[i][t];
113        }
114        fprintf(stdout, "(\%d): ftj0=\%f qtj0=\%f\n", t+1, ftj0, qtj0);
115        rtj= 1/qtj0;
116        stj=exp(-ftj0)/qtj0;
117        ftj1=log((rtj+Yt[t])/(stj+1));
118        qtj1=1/(rtj+Yt[t]);
119        fprintf(stdout, "(\%d): rtj=\%f stj=\%f ftj1=\%f qtj1=\%f\n", t+1, rtj,
               stj, ftj1, qtj1);
120        // calculando os momentos a posteriori
121        for (i=0; i<ncov; i++) {
122           aux[i]=0.0;
123           for (j=0; j<ncov; j++) {
124              aux[i]+=R[i][j][t] Ft[j][t];
125           }
126           fprintf(stdout, "(\%d): aux=\%f\n", t+1, aux[i]);
127           m[i][t]=a[i][t]+aux[i] ((ftj1-ftj0)/qtj0);
128           fprintf(stdout, "(\%d): mtj=\%f\n", t+1, m[i][t]);
129        }
130        for (i=0; i<ncov; i++) {
131           for (j=0; j<ncov; j++) {
132              aux2[i][j]=aux[i] Ft[j][t];
133              fprintf(stdout, "(\%d): aux2=\%f\n", t+1, aux2[i][j]);
134           }
135        }
136        for (i=0; i<ncov; i++) {
137           for (j=0; j<ncov; j++) {
138              aux3[i][j]=0.0;
139              for (k=0; k<ncov; k++) {
140                 aux3[i][j]+=aux2[i][k] R[k][j][t];
141              }
142              C[i][j][t]=R[i][j][t]-aux3[i][j] ((1-qtj1/qtj0)/qtj0);
143              fprintf(stdout, "(\%d): Ctj=\%f\n", t+1, C[i][j][t]);
144           }
145        }
146        // passando a equacao de evolucao
147        for (i=0; i<ncov; i++) {
148           a[i][t+1]=0.0;
149           for (j=0; j<ncov; j++) {
150              a[i][t+1]+=G[i][j] m[i][t];
```

```
151             }
152             fprintf(stdout, "(\%d): atj=\%f\n", t+1, a[i][t+1]);
153         }
154         for (i=0; i<ncov; i++) {
155             for (j=0; j<ncov; j++) {
156                 aux3[i][j]=0.0;
157                 for (k=0; k<ncov; k++) {
158                     aux3[i][j]+=G[i][k] C[k][j][t];
159                 }
160                 fprintf(stdout, "(\%d): aux3=\%f\n", t+1, aux3[i][j]);
161             }
162         }
163         for (i=0; i<ncov; i++) {
164             for (j=0; j<ncov; j++) {
165                 aux4[i][j]=0.0;
166 //              fprintf(stdout, "(\%d): ", t+1);
167                 for (k=0; k<ncov; k++) {
168                     aux4[i][j]+=aux3[i][k] G[k][j];
169 //                  fprintf(stdout, "aux3=\%f G=\%f ", aux3[i][k], G[k][j]);
170                 }
171                 fprintf(stdout, "(\%d): aux4=\%f\n", t+1, aux4[i][j]);
172             }
173         }
174         for (i=0; i<ncov; i++) {
175             for (j=0; j<ncov; j++) {
176                 R[i][j][t+1]=aux4[i][j]+(1/delta-1) C[i][j][t];
177                 fprintf(stdout, "(\%d): Rtj=\%f\n", t+1, R[i][j][t+1]);
178             }
179         }
180     }
181     return 1;
182 }
183
184 int DGLM::EscrevaResults(char  nomeArqSaida) {
185     fprintf(stdout, "DGLM::EscrevaResults\n");
186     int i, t;
187      FILE  arqSai;
188      arqSai = fopen(nomeArqSaida, "w");
189     for (t=0; t<tam; t++) {
190         for (i=0; i<ncov; i++) {
191             fprintf(stdout, "\%f ", m[i][t]);
192             fprintf(arqSai, "\%f ", m[i][t]);
193         }
194         fprintf(stdout, "\n");
195         fprintf(arqSai, "\n");
196     }
197     fclose(arqSai);
198     return 1;
199 }
200
201 #endif
```

Abordagem Bayesiana Dinâmica para o Ajuste do Modelo Exponencial por Partes

Paulo Cerqueira Jr.
Departamento de Estatística - ICEx - UFMG
31270-901 - Belo Horizonte - MG
E-mail: pauloest16@est.ufmg.br

Resumo

O modelo de Cox ou de riscos proporcionais é um dos modelos mais usados em análise de sobrevivência. Em tal modelo, é assumido que o risco de ocorrência do evento de interesse é constante em todo o tempo de acompanhamento. Em algumas situações, tal suposição é bem razoável e suficiente. Entretanto, é uma suposição muito restrita, uma vez que o efeito pode mudar com o tempo, havendo necessidade de ser incorporado na modelagem. Sob uma abordagem bayesiana, a modelagem dinâmica fornece uma estrutura em que o efeito da covariável possa variar no tempo, permitindo que os dados e as suposições *a priori* determinem as formas de variação, fornecendo uma possível solução para o problema.

4.1 Introdução

Em análise de sobrevivência a variável resposta, digamos T, é o tempo decorrido até a ocorrência de um evento de interesse que pode ser, por exemplo, o tempo até a morte de um paciente, a cura ou recidiva de uma doença, ou a ocorrência de uma falha em um equipamento (confiabilidade em engenharia), etc.

Uma característica peculiar dos dados de sobrevivência é a presença de censura, caracterizada pela observação incompleta da variável de interesse para alguns indivíduos do estudo, seja por iniciativa dos mesmos em sair do estudo, perda de contato durante o acompanhamento ou término do estudo sem ocorrência do evento, etc.

Tal censura pode ser classificado como, à direita, quando o tempo de ocorrência do evento de interesse é maior que o tempo registrado; à esquerda, quando o tempo registrado é maior que o tempo de ocorrência do evento de interesse e intervalar, quando ocorre quando

sabe-se apenas que o evento de interesse ocorreu em um certo intervalo de tempo. Neste projeto, optou-se em trabalhar sob a suposição de que as observações em estudo estão sob o mecanismo de censura a direita.

Com respeito à modelagem optou-se em usar o modelo exponencial por partes (MEP), (Ibrahim et al. 2001, Demarqui 2010, ver), que dentre muitos modelos, se tornou um dos mais populares na modelagem semiparamétrica de dados de sobrevivência, pois apesar de ser paramétrico, não requer suposições quanto à forma da função de taxa de falha, o que é diferente na classe dos modelos paramétricos (em que especificamos a forma fechada para a função de taxa de falha). Dessa forma, o MEP é considerado muito flexível.

Segundo Ibrahim et al. (2001) e Demarqui (2010), o MEP é caracterizado pela aproximação da função taxa de falha por segmentos de retas, cujos comprimentos são determinados por uma grade de pontos que divide o eixo dos tempos em um número finito (b) de intervalos.

Tal grade é definida como $\tau = \{a_0, a_1, ..., a_b\}$ e tem um papel fundamental nas estimativas dos parâmetros de interesse, bem como na qualidade do ajuste do modelo. Embora exista uma vasta literatura relacionada a este modelo (veja, Sahu et al. 1997, Breslow 1974, Kalbeisch 1973, Qiou et al. 1999, Ibrahim et al. 2001, já mencionados), a maioria trabalha com a grade τ de forma fixa.

Uma alternativa inovadora para a modelagem do MEP com grade aleatória é introduzida por Demarqui et al. (2008). Nesse projeto, a estrutura de agrupamento do modelo partição produto (MPP) proposto por Barry & Hartigan (1992) é utilizada para modelar diretamente a aleatoriedade de $\tau = \{a_0, a_1, ..., a_b\}$. Desta forma, a abordagem proposta por Demarqui et al. (2008) é completamente bayesiana e permite a estimação da grade que define o MEP, tornando a modelagem muito mais atrativa e elegante. Sob tal abordagem, a modelagem da grade é feita levando-se em conta a disposição dos tempos de falha sob o eixo do tempo e a existência de pelo menos um tempo de falha em cada intervalo induzido pela grade aleatória do MEP é garantida. Além disto, apesar de o número de parâmetros a ser estimados poder variar, esta forma de modelagem mantém fixado o número máximo de parâmetros do modelo.

4.2 Objetivos e Justificativa

Apesar da flexibilidade do modelo exponencial por partes com grade aleatória, o custo computacional relacionado ao tempo é muito grande, principalmente quando é realizada a atualização da grade a cada iteração do *Gibbs Sampler*. Há então a necessidade que tal modelo seja implementado em linguagens de de alto desempenho a modo deixar algoritmo mais rápido.

Desta forma, este projeto tem como objetivo e justificativa, a implementação através de uma função do algoritmo proposto por Barry & Hartigan (1992) em `C++` (uma vez que a atualização da grade é o setor mais lento) e chamá-la em `R` pelo pacote `Rcpp` e `RcppArmadillo`.

Uma vez que a proposta de modelagem é bem extensa comparando com o tempo para implementação, fora decidido em implementar o modelo mais simples, mas que é grande importância para a modelagem proposta. Assim, abaixo será descrito a modelagem dinâmica

proposto por Gamerman (1991).

4.3 Estimação via *Linear Bayes*

Na maioria dos modelos para dados de sobrevivência assume-se que os efeitos das covariáveis são fixos no risco de falha para um determinado indivíduo. Em algumas situações, tal suposição é bem razoável e suficiente. Entretanto, é uma suposição muito restrita, uma vez que o efeito pode mudar com o tempo, com por exemplo, dados de desemprego mudando ao longo do tempo (Gamerman & West 1987). A modelagem dinâmica fornece uma estrutura em que o efeito da covariável varie no tempo, permitindo que os dadas e as suposições *a priori* determinem as formas de variação, fornecendo uma possível solução para o problema.

Gamerman (1991) estende o modelo de riscos proporcionais, admitindo que os coeficientes das covariáveis variem em intervalos de tempo, através de um sistema de equações que fornecem uma descrição da evolução estocástica dos parâmetros ao longo do tempo. Tal abordagem utiliza uma análise seqüencial, baseada na fatorização da função de verossimilhança nos intervalos de tempo, assumindo a distribuição exponencial por partes para aos tempos observados.

Algumas definições preliminares, assume-se que T é uma variável aleatória, tal que, $T \sim MEP(\boldsymbol{\lambda}, \tau)$ com $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_b)$ e $\tau = \{a_0, a_1, \ldots, a_{b-1}, a_b\}$, para algum inteiro b e $a_0 = 0 \leq a_1 \leq \cdots \leq a_{b-1} \leq a_b = \infty$, e assim a função de taxa,

$$\lambda(t) = \lambda_j, \mathrm{t} \in I_i = (a_{j-1}, a_j], j = 1, \ldots, b. \tag{4.1}$$

A grade τ pode ser construída usando os tempos de falha observados, ou simplesmente selecionando-se os elementos de τ de forma arbitrária. Assumindo que $T \sim EPP(\boldsymbol{\lambda}, \tau)$, a função de sobrevivência e densidade condicional de T, dado $T \geq a_{j-1}$, que são necessários para a análise sequencial, são

$$S(t \mid T \geq a_{j-1}) = \exp[-\lambda_j(t - a_{j-1})], \mathrm{t} \in I_i, j = 1, \ldots, b, \tag{4.2}$$

$$f(t \mid T \geq a_{j-1}) = \lambda_j \exp[-\lambda_j(t - a_{j-1})], \mathrm{t} \in I_i, j = 1, \ldots, b. \tag{4.3}$$

Um vez que foram feitas as suposições acima, o modelo proposto em Gamerman (1991) será definido na subseção a seguir.

4.3.1 O Modelo

Seguindo um pouco a notação de Gamerman (1991), tome-se $T = (T_1, T_2, \ldots, T_n)$ como sendo um conjunto de tempos de sobrevivência independentes observados de um grupo de n indivíduos e $\tau = \{a_0, a_1, \ldots, a_{b-1}, a_b\}$, em que a_b é um valor maior que o maior tempo de sobrevivência observado. Dessa forma, a modelagem propõem os seguintes componentes

1. Equação de observação: $T_i \sim EPP(\boldsymbol{\lambda}_i, \tau)$, $\boldsymbol{\lambda}_i = (\boldsymbol{\lambda}_{i1}, \boldsymbol{\lambda}_{i2}, \ldots, \boldsymbol{\lambda}_{ib})$, $i = 1, \ldots, n$.

2. Guia de relação: $\log \lambda_{ij} = \boldsymbol{X}_i^{'}\boldsymbol{\beta}_j$, $j = 1, \ldots, b$.

3. Equação de evolução: $\boldsymbol{\beta}_j = \mathbf{G}_j\boldsymbol{\beta}_{j-1} + \mathbf{w}_j$, $\mathbf{w}_j \sim [\mathbf{0}, \mathbf{W}]$.

em que $\boldsymbol{X}^{'} = (\mathbf{1}, X_{i,1}, \ldots, X_{i,p})$ com $X_{i,k}$, $k = 1, \ldots, p$, sendo o valor da $k-$ésima covariável para o $i-$ésimo indivíduo; $\boldsymbol{\beta}_j^{'} = (\boldsymbol{\beta}_{j0}, \boldsymbol{\beta}_{j1}, \ldots, \boldsymbol{\beta}_{jp})$ é o conjunto de coeficientes de regressão; $\mathbf{G}_j$ é a matriz do sistema de evolução, quase sempre dependente de b_j (comprimento do intervalo j). Frequentemente a, $\mathbf{G}_j = \mathbf{I}_p$, com $\mathbf{I}_p$ sendo uma matrix identidade p dimensional, nesta tese não será diferente, e w_j é o erro acumulado sob o intervalo I_j, quase sempre assumi-se que $\mathbf{w}_j \sim [\mathbf{0}, \mathbf{W}]$, ou seja, $\mathbf{0}$ é um vetor de médias e $\mathbf{W}$ é uma matrix diagonal, cuja os valores são chamados de parâmetros de suavização, definidos *ad hoc*.

Uma vez que o interesse é de usar modelos que captem o efeito da covariável ao longo do tempo, a função de taxa de falha fica na forma com encontrado "guia de relação´´, em que $\lambda_{ij} = \exp\{X_i^{'}\boldsymbol{\beta}_j\}$, $j = 1, \ldots, b$. Como mencionado anteriormente, a informação dos dados contidos na função de verossimilhança é dividida em intervalos de tempos. A ideia é chamada de *Fatorização temporal da função de verossimilhança*. Assim a função de verossimilhança precisa ser adaptada, para levar em consideração somente as informações contidas intervalo I_j, induzidos pela partição τ. Então considerando a informação obtida em cada intervalo de tempo, toma-se L_i como os fatores da função de verossimilhança, que possui a seguinte forma,

$$\prod_{j=1}^{n_j} (\lambda_{ij})^{\delta_{ij}} \exp\left[-\lambda_{ij}(t_{ij} - a_{j-1})\right] \tag{4.4}$$

em que n_j é o numero de indivíduos que estão sob risco no início de cada intervalo I_j, δ_{ij} é o indicador de falha para o $i-$ésimo indivíduo no $j-$ésimo intervalo, e

$$t_{ij} = \begin{cases} a_{j-1}, & \text{se} \quad t_i < a_{j-1} \\ t_i, & \text{se} \quad t_i \in I_j \\ a_j, & \text{se} \quad t_i > a_j, \end{cases} \tag{4.5}$$

Com o modelo definido, o processo inferêncial dos parâmetros de interesse serão descritos em seguida.

4.3.2 Inferência dos Parâmetros via Análise Sequencial

A inferência é baseada em dois momentos, primeiramente é realizada uma atualização *on line*, ou seja, cada $\boldsymbol{\beta}_j$ será estimdo baseando-se somente em D_j, que representa toda a informação até o intervalo j e assim, é obtida a distribuição *on line* $(\boldsymbol{\beta}_j \mid D_j)$. Após a atualização em todos os intervalos é realizado uma suavização das estimativas *on line* utilizando uma análise recursiva.

4.3.3 Atualização *on Line*

Tomando o tempo a_{j-1}, assuma que a distribuição *a posteriori* para o parâmetro $\boldsymbol{\beta}_{j-1}$ é dada por $[\boldsymbol{\beta}_{j-1}|D_{j-1}] \sim [\mathbf{m}_{j-1}, \mathbf{C}_{j-1}]$, em que D_{j-1}, que representa toda a informação até o tempo a_{j-1}. Uma vez que $\mathbf{G}_j$ como uma matrix identidade, a análise sequencial basea-se em um passeio aleatório de primeira ordem *a priori*, e assim a distribuição condicional *a priori* para $\boldsymbol{\beta}_j$ é dada por $[\boldsymbol{\beta}_j|D_{j-1}] \sim [\mathbf{a}_j; \mathbf{P}_j] = [\mathbf{m}_{j-1}; \mathbf{C}_{j-1} + \mathbf{W}]$.

Os momentos *a posteriori* de $[\boldsymbol{\beta}_j \mid D_j]$ pode ser obtida ao processar a informação de $i-$ésimo indivíduo sob risco dentro de cada intervalo j de forma sequencial. Como citado anteriormente, assuma que exista n_j indivíduos sob risco no início do $j-$ésmo intervalo a_{j-1} e seja $D_{i-1;j-1}$, para $i = 1, 2, \ldots, n_j$ o conjunto de toda informação disponível até o I_{j-1}, juntamente com a informação processada associada com os primeiros $i-1$ indivíduos que ainda estão sob risco no início do intervalo I_j. A distribuição condicional *a priori* de $\boldsymbol{\beta}_j$ antes de processar a informação do $i-$ésimo indivíduo pode ser denotado por $[\boldsymbol{\beta}_j|D_{i-1;j-1}] \sim [\mathbf{a}_{ij}; \mathbf{P}_{ij}]$. Uma vez que $\log \lambda_{ij} = \boldsymbol{X}_{ij}^{'}\boldsymbol{\beta}_j$, a distribuição conjunta condicional *a priori* de $[\boldsymbol{\beta}_j, \log \lambda_{ij}]$ é dada por

$$\left[\begin{pmatrix} \boldsymbol{\beta}_j \\ \\ \log \lambda_{ij} \end{pmatrix} \mid D_{i-1;j-1} \right] \sim \left[\begin{pmatrix} \mathbf{a}_{ij} \\ \\ f_{ij} \end{pmatrix}, \begin{pmatrix} \mathbf{P}_{ij} & \mathbf{s}_{ij} \\ \\ \mathbf{s}_{ij}^{'} & q_{ij} \end{pmatrix} \right],$$

em que $f_{ij} = \mathbf{X}_{ij}^{'}\mathbf{a}_{ij}$, $\mathbf{a}_{ij} = \mathbf{P}_{ij}\mathbf{X}_{ij}$, $q_{ij} = \mathbf{X}_{ij}^{'}\mathbf{s}_{ij}$ e λ_{ij} é a função de taxa de falha para o $i-$ésimo indivíduo no $j-$ésimo intervalo. Inicialmente tome $\mathbf{a}_{0j} = \mathbf{a}_j$, $\mathbf{P}_{0j} = \mathbf{P}_j$ e $D_{0;j-1} = D_{j-1}$.

A distribuição Gama é assumida com distribuição *a priori* para a função de taxa de falha λ_{ij}, e como há conjugação com a função de verossimilhança em (4.4), a ditribuição *a posteriori* também terá distribuição Gama, podendo ser resolvida de forma analítica. Com os momentos da distribuição *a posteriori* de λ_{ij} e a distribuição condicional conjunta *a priori* definida anteriormente, a distribuição *a posteriori* $[\boldsymbol{\beta}_j|D_{i;j-1}] \sim [\mathbf{m}_{ij}; \mathbf{C}_{ij}]$ é estimado pelo método *Linear Bayes* descrito em West et al. (1985), então

$$\mathbf{m}_{ij} = \mathbf{a}_{ij} + \frac{\mathbf{s}_{ij}}{q_{ij}} \ln \left\{ \frac{1 + q_{ij}\delta_{ij}}{1 + q_{ij}(t_{ij} - a_{j-1}) \exp(f_{ij})} \right\}, \tag{4.6}$$

$$\mathbf{C}_{ij} = \mathbf{P}_{ij} - \frac{\delta_{ij}}{1 + q_{ij}\delta_{ij}} \mathbf{s}_{ij}\mathbf{s}_{ij}^{'}, \tag{4.7}$$

em que $\delta_{ij} = 1$ se o $i-$ésimo indivíduo sob risco no intervalo I_j foi evento durante o intervalo j. Os passos de atualização dentro de cada intervalo e evolução são descritos abaixo

$$[\boldsymbol{\beta}_0|D_0] \to E \to [\boldsymbol{\beta}_1|D_0] \to A \to [\boldsymbol{\beta}_1|D_1] \to E \to [\boldsymbol{\beta}_2|D_1] \to A \to [\boldsymbol{\beta}_2|D_2] \ldots [\boldsymbol{\beta}_b|D_b].$$

4.3.4 Atualização via Distribuição Suavizada

Na atualização Suavizada a estimativa é baseada na informação obtida por todos os intervalos, representados por D_b. Assim os momentos de $[\boldsymbol{\beta}_j|D_b]$,

$$\mathbf{m}_j^b = \mathbf{m}_j + \mathbf{C}_j\mathbf{P}_{j+1}^{-1}[\mathbf{m}_{j+1}^b - \mathbf{a}_j]$$
$$\mathbf{C}_j^b = \mathbf{C}_j - \mathbf{C}_j\mathbf{P}_{j+1}^{-1}[\mathbf{P}_{j+1} - \mathbf{C}_{j+1}^b]\mathbf{P}_{j+1}^{-1}\mathbf{C}_j.$$

em que, $\mathbf{m}_b^b = \mathbf{m}_b$ e $\mathbf{C}_b^b = \mathbf{C}_b$.

4.4 Resultados Pretendidos

Pretende-se reduzir ao máximo o tempo computacional do modelo exponencial por partes com grade aleatória, principalmente na atualização da grade. E assim, poder usar o modelo de forma mais eficiente, como por exemplo, através de um pacote em R.

4.5 Resultados Parciais

Usando uma interface no software R via pacote Rcpp e RcppArmadillo:

```
1
2  //#########################################################
3  //## Dynamic.cpp                                         ##
4  //#########################################################
5  //## Estima os coeficientes suavizados via modelo        ##
6  //## Dinamico Gamermam(1991)                             ##
7  //#########################################################
8  //## Argumentos:                                         ##
9  //#########################################################
10 //# data: Matriz com todas as informações               ##
11 //# temp: tempo;                                        ##
12 //# id_fail: indicador de falha;                        ##
13 //# matrix_cov: Matriz de covariaveis;                  ##
14 //# grid_vet: Vetor de valores da grade dos tempos;     ##
15 //# dist_W: Matriz de variancia e cov do erro;          ##
16 //#########################################################
17
18 // [[Rcpp::depends(RcppArmadillo)]]
19 #include <RcppArmadillo.h>
20 using namespace arma;  // Biblioteca Armadillo para operacoes matriciais.
21 using namespace Rcpp;  // Biblioteca que realiza o link com o R.
22
23
24 // [[Rcpp::export]]
25 List gamerman91C(mat data, vec temp, vec idfail, vec postime,  mat matrixCov, vec gridTime, mat distW)
26
27 // Fim do codigo.
```

Listing 4.1: Implementação do modelo de dinâmico via função Gamerman91

4.6 Resultados Esperados

O ajuste do modelo dinâmico foi aplicado à dados de pacientes com câncer de mama, em que foram comparados dois grupos, os pacientes que realizaram somente quimioterapia e os que fizeram quimioterapia+radioterapia (dados utilizados em Gamerman 1991). Resumindo, os resultados esperados é do gráfico do efeito da covariável tipo de tratamento do câncer. Tal gráfico (resultado de implementação em R) é apresentado na figura a seguir.

A linha horizontal sólida em vermelho, representa a estimativa do modelo de Cox para o efeito do tratamento com $\beta_1 = 0.065$, constante em todo o tempo de acompanhamento. Observe que o efeito do tipo de tratamento ajustado pelo modelo dinâmico, cresce no início e em seguida cai, nos apresentando que o efeito muda com o tempo.

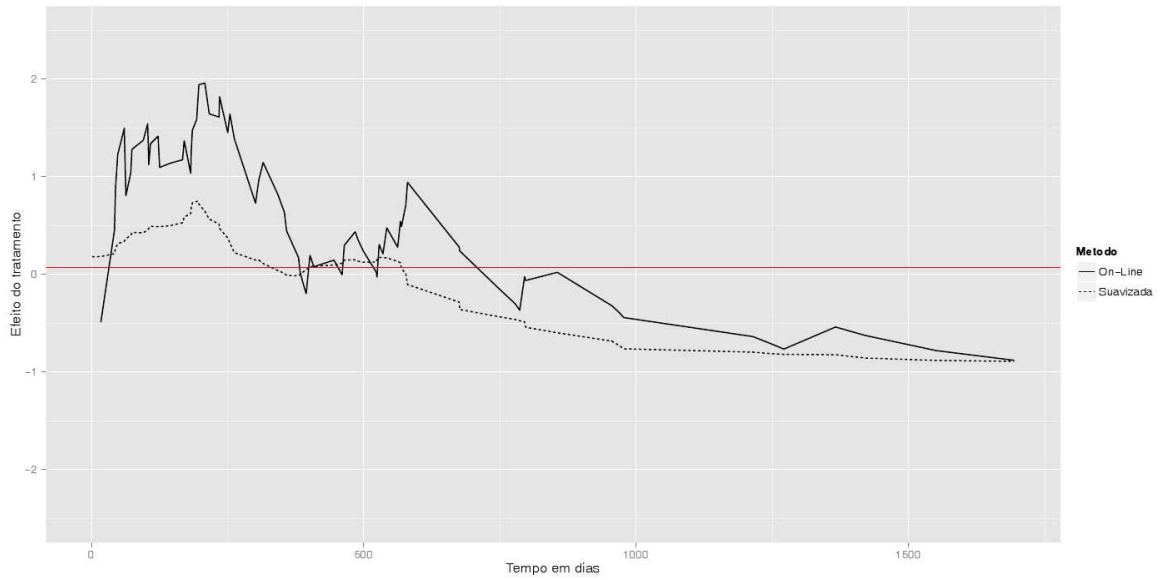

Figura 4.1: Efeito do tipo de tratamento no tempo das pacientes com câncer (ajuste realizado em R).

4.7 Conclusões Parciais

Ao termino deste relatório, ainda não foi possível apresentar a Figura (4.1) usando os resultados da implementação em C++. No entanto, trabalho de programação permitiu, obter novos aprendizados com respeito aos pacotes Rcpp e RcppArmadillo, sobre seu estilo de sintax (mais importante) e compilação dos códigos.

Referências Bibliográficas

Barry, D., & Hartigan, J. A. (1992). Product partition models for change point problems. *Annals of Statistics*, *20*, 260–279.

Breslow, N. E. (1974). Covariance analysis of censored survival data. *Biometrics*, *30*, 89–99.

Demarqui, F. N. (2010). *Uma classe mais flexível de modelos semiparamétricos para dados de sobrevivência*. Tese de doutorado, UFMG.

Demarqui, F. N., Loschi, R. H., & Colosimo, E. A. (2008). Estimating the grid of time-points for the piecewise exponential model. *Lifetime Data Analysis*, *14*, 333–356.

Gamerman, D. (1991). Dynamic bayesian models for survival data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *40, No. 1*, 63–79.

Gamerman, D., & West, M. (1987). An application of dynamic survival models in unemployment studies. *The Statistician*, *36*, 269–274.

Ibrahim, J. G., Chen, M. H., & Sinha, D. (2001). *Bayesian Survival Analysis*, vol. 1. Springer-Verlag.

Kalbeisch, R. L., J. D. e Prentice (1973). Marginal likelihoods based on Cox´s regression and life models. *Biometrika*, *60*, 267–278.

Qiou, Z., Ravishanker, N., & Dey, D. K. (1999). Multivariate survival analysis with positive stable frailties. *Biometrics*, *55*, 585–590.

Sahu, S. K., Dey, D. K., Aslanidou, D. K., & Sinha, D. (1997). A Weibull regression model with gamma frailties for multivariate survival data. *Lifetime data analysis*, *3*, 123–137.

West, M., Harrison, P. J., & Migon, H. S. (1985). Dynamic generalized linear models and Bayesian forecasting. *Journal of the American Statistical Association*, *80, No. 389*.

A. Códigos

```
1
2  //#########################################################
3  //## Dynamic.cpp                                         ##
4  //#########################################################
5  //## Estima os coeficientes suavizados via modelo        ##
6  //## Dinamico Gamermam(1991)                             ##
7  //#########################################################
8  //## Argumentos:                                         ##
9  //#########################################################
10 //# data: Matriz com todas as informações               ##
11 //# temp: tempo;                                        ##
12 //# id_fail: indicador de falha;                        ##
13 //# matrix_cov: Matriz de covariaveis;                  ##
14 //# grid_vet: Vetor de valores da grade dos tempos;     ##
15 //# dist_W: Matriz de variancia e cov do erro;          ##
16 //#########################################################
17
18 // [[Rcpp::depends(RcppArmadillo)]]
19 #include <RcppArmadillo.h>
20 using namespace arma;  // Biblioteca Armadillo para operacoes matriciais.
21 using namespace Rcpp;  // Biblioteca que realiza o link com o R.
22
23
24 // [[Rcpp::export]]
25 List gamerman91C(mat data, vec temp, vec idfail, vec postime,  mat matrixCov, vec gridTime, mat distW)
26 {
27
28  int nsize = matrixCov.n_rows;         // Sample size
29  int nintervals = gridTime.n_elem-1; // Number of intervals
30
31  // Calculando o numero de indivíduos sob risco:
32
33  // TRABALHANDO AINDA!!
34
35  //#########################################################
36  //### ATUALIZANDO OS MOMENTOS DA DISTRIBUICAO ON-LINE ###
37  //#########################################################
38
39  //--- Chute inicial para beta \beta_{j-1}|D_{j-1}:
40
41  vec  m = zeros(matrixCov.n_cols);
42  mat  C = zeros(matrixCov.n_cols,matrixCov.n_cols);
43
44  //--- Valores a priori para B_{j}|D_{j-1}:
45
46  vec aij = m;
47  mat Pij = C;
48
49  //--- Matrizes e vetores de armazenamento:
50
51  mat  betaj   = zeros( matrixCov.n_cols, nintervals);
52  cube vbetaj = zeros(matrixCov.n_cols,matrixCov.n_cols,nintervals);
53
54  //--- Definindo algumas quantidades:
55
56  double fij, qij, tij, bnum, bdeno, vdeno;
57  vec mij = m;
58  mat sij, Cov,Cij;
59
60  for( int j=0; j<nintervals; j++ ) // Inicio da evolucao entre os intervalos:
61  {
62
63 //  y <- under.risk[[j]][,1]
64 //  fail <-  under.risk[[j]][,2]
65 //  X_cov <- as.matrix(under.risk[[j]][,3:4])
66 //  delta_ij <- calc_delta_ij(y, fail, grid.time[j], grid.time[j+1])
67 //  aux.grid.time <- grid.time[-1] # Grade sem o "0".
68
69     for( int i=0; i<nsize; i++)  //Início da atualização dentro dos intervalos
70     {
71         //--- Calculando algumas quantidades:
72
73         Cov = matrixCov.row(i);
74
75         fij = as_scalar( Cov*aij );      // Média de log\lambda_ij|Dj-1
76         sij = Pij*Cov.t();               // Covariância entre Bj|Dj-1 e log\lambda_ij|Dj-1
77         qij = as_scalar( Cov*sij );      // Variância de log\lambda_ij|Dj-1
78
79       //-- Betaj:
80
81         //- Média à posteriori:
82
83           bnum = 1 + ( qij*idfail(i) );
```

```
          tij = std::min(temp(i),gridTime(j+1));
          bdeno = 1 + (qij*(tij-gridTime(j))*std::exp(fij));
          mij = aij + sij*((1/qij )*std::log( bnum/bdeno ));

        //- Variância à posteriori:

          vdeno = 1 + ( idfail(i)*qij );
          Cij = Pij - ( (sij*sij.t())*( idfail[i]/vdeno ) );

      //--- Recursividade:

        aij = mij;
        Pij = Cij;
    } // Fim da atualização dentro dos intervalos

    //--- Guardando as estimativas:

    betaj(,j)   = mij;
    vbetaj(,,j) = Cij;

    //--- Evolucao:

    aij = mij     // Média de Bj|Dj-1
    Pij = Cij + distW // Variância de Bj|Dj-1

 } //# fim da evolução entre os intervalos

 // Organizando os resultados na lista:

 List result;
 result[´Sample_size´]=nsize;
 result[´Sample_size´]=nintervals;
 result[´Betaj´]   = betaj;
 result[´V_Betaj´]   = vbetaj;

 return result;
}
// Fim do codigo.
```

Listing 4.2: Implementação do modelo de dinâmico via função Gamerman91C

```
 ##########################################################
 # Autor: Paulo Cerqueira Jr.                             #
 # Código: Ajuste do MEP dinamica sob abordagem           #
 # Bayesiana (Gamerman, 1991)                             #
 # Início: 05/10/2014                                     #
 # Fim:    20/10/2014                                     #
 ##########################################################


#--- Carregando alguns pacotes:

require(survival)
library(MASS)
library(compiler)
library(Rcpp)
library(RcppArmadillo)

#--- Fixando a semente de dados:

set.seed(1234)

#--- Lendo os dados (Gamermam, 1991):

dados <- read.table(´dados_dani_91.txt´, h=T)
dados <- dados[order(dados$t),]
tempo <- dados$t
cens  <- dados$cens
n     <- length(tempo)
X     <- matrix(c( rep(1,n), dados$tr), ncol=2, nrow=n)
dados <- as.matrix(dados)

#--- Estipulando os valores da grade:

grid.time <- c(0, unique(tempo[cens==1]),max(unique(tempo[cens==1]))*10)
pos.tempo <- as.numeric(cut(tempo, grid.time))
W <- diag(0, ncol=dim(X)[2], nrow=dim(X)[2])

# Testando o modelo de DANI91 em C++:

fit <- gamerman91C( dados, tempo, cens, pos.tempo,  X, grid.time, W)

# Fim do código
```

Listing 4.3: Código em R que se relaciona com o C++ via função Gamerman91C

Simulação de Rebanho de Bovinos de Corte

Tarciso Pereira Martins Junior
Departamento de Estatística - ICEx - UFMG
31270-901 - Belo Horizonte - MG
E-mail: tarciso.junior@gmail.com

Resumo

Neste trabalho, apresentam-se simulações de valores genéticos e fenotípicos para uma característica qualquer (peso) e acasalamentos para produção de novas gerações de animais.

5.1 Introdução

De acordo com Pereira (2012), o fenótipo não é o resultado somente da constituição genética do indivíduo, mas também da interação dos seus genes com os vários efeitos não genéticos ou de ambiente. Não faz sentido perguntar se uma característica é hereditária ou ambiente. Para que os genes possam provocar o desenvolvimento de uma característica é preciso que disponham de ambiente adequado. Por outro lado, as modificações que o ambiente pode causar no desenvolvimento de uma característica são limitadas pelo genótipo do indivíduo. É preciso reconhecer, todavia, que a variabilidade observada em algumas características pode ser causada pelas diferenças gênicas entre os diversos indivíduos. A variabilidade de algumas características pode ser consequência das diferenças nos ambientes aos quais os indivíduos foram expostos. Quanto mais o ambiente influencia nas ações dos genes, menos exata será a estimativa do genótipo do indivíduo. Nas características econômicas (peso e ganho em peso, produção de leite, número de leitões por leitegada, produção de ovos etc.) o progresso que pode ser alcançado está na dependência da melhor ou pior precisão em avaliar os genótipos, tendo por base o fenótipo dos indivíduos. Métodos estatísticos apropriados permitem estimar o quando da variação fenotípica é devida às diferenças genéticas entre os indivíduos e o quanto é devido às diferenças de natureza ambiente.

Nesse trabalho, será desconsiderada a interação herança $\times$ meio, e a variância fenotípica de uma população é determinada por:

$$\sigma_P^2 = \sigma_H^2 + \sigma_E^2,$$

em que:

- σ_P^2 = variância fenotípica;
- σ_H^2 = variância genética hereditária;
- σ_E^2 = variância devida a efeitos de ambiente.

5.1.1 Objetivos

Pretende-se neste trabalho desenvolver um sistema em c++ para simular uma população. A população deverá conter os dados individuais de valor genético, valor de efeito de ambiente, valor fenotípico, identificador do pai, identificador da mãe e coeficiente de geração.

5.1.2 Organização

Este texto está organizado da seguinte forma: Na seção 5.2, descrevem-se os materiais e métodos utilizados. Os resultados são discutidos na seção 5.3. As conclusões finais apresentadas na seção 5.4 encerram o texto.

5.2 Materiais e Métodos

Espera-se construir um aplicativo para processar as entradas e saídas abaixo descritas

5.2.1 Entradas

- Tamanho do rebanho a ser gerado
- Quantidade de ciclos de acasalamento
- Quantidade de machos por ciclo
- Quantidade de femeas por ciclo
- Média da distribuição normal para o valor genético
- Desvio Padrao da distribuição normal para o valor genético
- Média da distribuição normal para o efeito de ambiente
- Desvio Padrao da distribuição normal para o efeito de ambiente

5.2.2 Processamento

A rotina proposta deverá gerar uma listagem de indivíduos com os atributos abaixo descritos, realizar rotinas de seleção e acasalamento entre esses indivíduos, incorporando os indivíduos gerados à população inicial.

Valor genético (H): determinado por uma variável aleatória que segue uma distribuição normal, com média e desvio padrão informados na entrada,

$$H \sim N(\mu_h, \sigma_h).$$

Valor do efeito de ambiente (E): determinado por uma variável aleatória, com distribuição normal, com média e desvio padrão informados na entrada,

$$E \sim N(\mu_e, \sigma_e).$$

Valor Fenotípico (P): definido pela soma dos valores genéticos e do efeito de ambiente,

$$P = H + E$$

Sexo: definido aleatoriamente, assume os valores 1 para macho e 2 para fêmea.

Fator de geração: fator de geração resultante da média dos pais + 1,

$$FG = (Pai.FG + Mae.FG/2) + 1$$

5.2.3 Saída

- Listagem do rebanho, com os valores genéticos, efeito de ambiente, fenotípico, sexo, e fator de geração

5.2.4 Classes Propostas

Para o desenvolvimento do programa, serão desenvolvidas duas classes na linguagem de programação c++: Indivíduo, contendo atributos relativos à identificação do indivíduo, seus valores genéticos, efeito de ambiente e fenotípicos, sobrecarga de operadores de adição e atribuição, responsáveis pela atribuição de valores dos novos indivíduos criados durante os processos de simulação de acasalamentos e geração da progênie, e uma classe denominada População, contendo uma lista de indivíduos e os métodos de seleção, acasalamento e geração da progênie. Também foi necessário implementar um algoritmo de ordenação. Em particular utilizou-se o algoritmo da bolha (Nelder & Wedderburn 1972), para classificação e seleção dos indivíduos eletivos às rotinas de acasalamentos. A Figura 5.1 descreve a declaração da

classe indivíduo. Essa classe contém os atributos de cada indivíduo e uma sobrecarga de operadores, para soma e atribuição, possibilitando a operação

$$objFilho = objPai + objMae$$

durante o processamento. A Figura 5.2 descreve a classe população, que é uma lista de indivíduos com os métodos de geração da população base, geração das progênies, seleção e acasalamentos. Um exemplo de chamada das classes pode ser visto na Figura 5.3. Na classe população, são utilizados os códigos escritos na sobrecarga de operadores da classe Individuo, onde são gerados os filhos, conforme descrito na Figura 5.4.

```
1  class Individuo{
2      public:
3      int idIndividuo ;
4      int idPai, idMae;
5      int sexo;
6      float vg, ve, vf, geracao;
7
8
9      public:
10     Individuo(void);
11     Individuo(const Individuo &pIndividuo);
12     Individuo(int id);
13     ~Individuo(void);
14
15     Individuo operator + (const Individuo &p);
16     Individuo &operator = (const Individuo &pIndividuo);
17
18     int getId();
19     int getSexo();
20     float getValorGenetico();
21     float getValorAmbiente();
22     float getValorFenotipico();
23     int setSexo(int valor);
24     int setValorGenetico(float valor);
25     int setValorAmbiente(float valor);
26
27 };
```

Figura 5.1: Classe para implementação em C++ do objeto *individuo*

5.3 Resultados e Discussão

Como resultado do processamento, tem-se uma lista de indivíduos, com um identificador, seus valores genéticos e fenotípicos, sexo, identificadores do pai e da mãe e do coeficiente de geração, que é definido no momento da geração do indivíduo, pela média dos valores de

```
class Populacao {
    public:
    Individuo *pop;
    int tam, tamBase;
    float mediaG, dpG, mediaE, dpE;
    public:
    Populacao(void);
    Populacao(int tamanho);
    ~Populacao(void);

    int geraPopulacaoBase(float media, float desvioPadrao);
    int geraPopulacaoBase(int tamanho, float mediaVG, float dpVG, float mediaEA, float dpEA);
    int ordena(int campoOrdem);
    int ordena_desc(int campoOrdem);
    int geraProgenie(int nMachos, int nFemeas, int nGeracoes, int selecao);
    void imprime(FILE *saida);
    void imprime_csv(FILE *saida);

    Individuo getIndividuo(int pos);
    Populacao filtraValor(int campo, int valor);
    Populacao filtraValor(int campo, int valor, int qtde);

};
```

Figura 5.2: Classe para implementação em C++ do objeto *populacao*

```
#include <stdio.h>
#include <math.h>
#include "populacao.hpp"
#include "populacao.cpp"

int main(int argc, char **argv){
    int tamBase = (argc>2)?atoi(argv[1]): 30;
    int nGeracoes = (argc>3)?atoi(argv[2]): 5;
    int nMachos = (argc>4)?atoi(argv[3]): 2;
    int nFemeas = (argc>5)?atoi(argv[4]): 4;
    int tamTotal = int(tamBase + (nFemeas * nGeracoes));
    int selecao = (argc>6)?atoi(argv[5]): 3; // 1-vg; 2-ve; 3-vf
    float mVGBase = (argc>7)?atof(argv[6]):0;
    float dpVGBase = (argc>8)?atof(argv[7]):1;
    float mVEBase = (argc>9)?atof(argv[8]):0;
    float dpVEBase = (argc>10)?atof(argv[9]):1;

    Populacao pop(tamTotal);

    pop.geraPopulacaoBase(tamBase,mVGBase,dpVGBase,mVEBase,dpVEBase);

    pop.geraProgenie(nMachos,nFemeas,nGeracoes,selecao);

    pop.ordena(0);
    pop.imprime_csv(stdout);

    return(0);
}
```

Figura 5.3: Código para utilização em C++ das classes *individuo* e *populacao*

```
187 int Populacao::geraProgenie(int nMachos, int nFemeas, int nGeracoes, int selecao){
188
189     int pos = tamBase;
190
191     std::default_random_engine generator;
192     std::normal_distribution<float> environment(mediaE, dpE);
193     std::normal_distribution<float> mendeliana(0,dpE/2);
194
195     for (int g=0; g<nGeracoes; g++){
196         ordena_desc(selecao);
197         ordena_desc(5); // ordena por geracao
198         Populacao popM = filtraValor(4,1,nMachos);
199         Populacao popF = filtraValor(4,2,nFemeas);
200         popM.ordena_desc(selecao);
201         popF.ordena_desc(selecao);
202         int aux;
203         for (int i=0; i<nFemeas; i++){
204             aux = rand() % nMachos;
205             Individuo filho = popM.getIndividuo(aux) + popF.getIndividuo(i);
206             filho.idIndividuo = pos+1;
207             filho.setValorGenetico(filho.vg + mendeliana(generator));
208             filho.setValorAmbiente(environment(generator));
209             pop[pos++] = filho;
210         }
211     }
212     return 1;
213 }
```

Figura 5.4: Código da geração de progênie na classe *populacao*

geração do pai e da mãe adicionado do valor 1. Espera-se, com a análise da distribuição dos valores, observar e estudar o variação dos valores através das gerações, de acordo com o critério de seleção utilizado.

5.4 Conclusões e Observações Finais

A disponibilização desse trabalho visa facilitar a geração de populações simuladas para estudos de melhoramento animal por geneticistas e pesquisadores sem experiência com programação de computadores.

Esse trabalho tem função somente acadêmica, com o objetivo de demonstrar o processo simplificado de simulação de rebanho e expressão de características. Para que esse programa possa simular uma situação real, deve-se gerar arquivos com mais de uma característica, cada uma composta pelos valores genéticos e de ambiente, e considerar que pode haver correlação entre características que deverão ser explicitadas nas entradas.

Para melhorias futuras espera-se o tratamento para mais de uma característica, o desenvolvimento de uma interface guiada de uso, e disponibilização das classes compiladas para uso por terceiros.

Referências Bibliográficas

Pereira, J. C. C. (2012), *Melhoramento Genético Aplicado à Produção Animal*, FEPMVZ Editora, Escola de Veterinária da UFMG, Belo Horizonte, MG, capítulo Herança e Meio.

Ziviani, N. (1993), *Projeto de Algoritmos - Com Implementações em Pascal e C*, Pioneira, São Paulo, Brasil.

A. Exemplo de Saída

ID	SX	VG	VE	VF	idPai	iidMae	Geracao
1	2	-1.829	-1.499	-3.328	0	0	1.000
2	2	-16.302	0.175	-16.128	0	0	1.000
3	1	6.895	0.395	7.290	0	0	1.000
4	1	2.837	-2.370	0.467	0	0	1.000
5	2	0.504	-0.636	-0.132	0	0	1.000
6	1	-7.900	-0.171	-8.071	0	0	1.000
7	1	6.938	0.094	7.032	0	0	1.000
8	1	3.010	1.204	4.215	0	0	1.000
9	1	19.648	0.175	19.823	0	0	1.000
10	1	15.031	0.274	15.306	0	0	1.000
11	2	-1.918	1.317	-0.601	0	0	1.000
12	2	-5.052	0.681	-4.371	0	0	1.000
13	2	-2.075	0.860	-1.215	0	0	1.000
14	2	1.231	0.294	1.525	0	0	1.000
15	2	-14.038	-0.792	-14.829	0	0	1.000
16	2	-40.085	-0.922	-41.008	0	0	1.000
17	2	0.111	-2.004	-1.893	0	0	1.000
18	1	-10.777	0.180	-10.597	0	0	1.000
19	2	-13.419	-2.418	-15.838	0	0	1.000
20	1	-0.197	0.015	-0.183	0	0	1.000
21	1	8.001	-1.499	6.502	10	14	2.000
22	2	4.500	0.175	4.675	3	5	2.000
23	1	2.548	0.458	3.006	3	11	2.000
24	1	6.327	0.189	6.516	10	13	2.000
25	2	3.722	0.044	3.766	7	17	2.000
26	1	1.348	-0.664	0.683	3	1	2.000
27	1	4.671	1.731	6.403	10	12	2.000
28	1	-5.599	-0.765	-6.364	8	15	2.000
29	2	-2.911	0.094	-2.817	3	19	2.000
30	1	1.745	1.204	2.950	9	2	2.000
31	2	3.655	1.681	5.336	23	22	3.000
32	2	3.851	-0.702	3.149	27	25	3.000
33	1	1.977	-0.863	1.114	23	14	2.500
34	2	4.390	1.186	5.576	21	5	2.500
35	1	0.974	0.943	1.916	23	11	2.500
36	1	1.639	0.828	2.467	27	13	2.500
37	2	2.096	0.860	2.957	27	17	2.500
38	2	0.985	0.294	1.279	27	29	3.000
39	2	0.126	-0.132	-0.006	23	1	2.500
40	1	-2.488	1.577	-0.911	30	12	2.500
41	2	2.618	-1.047	1.571	36	34	3.500
42	1	4.530	-0.442	4.088	24	31	3.500
43	1	3.716	0.010	3.727	33	22	3.250
44	1	5.008	0.920	5.928	24	25	3.000
45	1	-0.321	-0.004	-0.325	40	32	3.750
46	2	1.958	0.329	2.286	36	37	3.500
47	2	3.568	-2.418	1.149	24	14	2.500
48	1	3.535	0.015	3.550	24	38	3.500
49	2	1.101	-2.366	-1.264	36	39	3.500
50	1	-0.585	1.979	1.394	33	29	3.250

Índice Remissivo

Almeida, J. A., 19

Cerqueira Jr., P., 33

Martins Junior, T. P., 45

Oliveira, G. L., 1

Rocha, J. C., 13