

Rotinas em R para Técnicas de Suavização por Núcleos Estimadores

Lupércio França Bessegato Gregorio Saravia Atuncar
Luiz Henrique Duczmal

Departamento de Estatística
Universidade Federal de Minas Gerais
30123-970 Belo Horizonte - MG

e-mails: lupercio@est.ufmg.br,
duczmal@est.ufmg.br, gregorio@est.ufmg.br

Resumo

O método do núcleo estimador, dentre outros, tem sido largamente utilizado em estimação funcional da função de densidade e de distribuição em suas várias aplicações. A utilização do método é sensível à escolha do parâmetro de suavização. Neste trabalho é apresentado um conjunto de funções em R para estimação da janela ótima de acordo com método ‘plug-in’ que utiliza de função característica empírica no tratamento dos dados.

Palavras-chave: Núcleo-estimador, Método da ‘plug-in’, Escolha da Janela Ótima, Função Característica.

Sumário

1	Introdução	3
2	Estimação Funcional	5
2.1	Estimação da Função de Densidade	5
2.1.1	A Janela Ótima e as Propriedades do Estimador	7
2.1.2	Estimador ‘Plug-in’	9
2.2	Estimação da Função de Distribuição	11
2.2.1	A Janela Ótima e as Propriedades do Estimador	12
2.2.2	Estimador ‘Plug-in’	12
2.3	Estimação da Função de Intensidade	13
2.3.1	A Escolha da Janela Ótima	14
2.4	Problemas de Fronteira	14
3	Funções em R	16
3.1	Implementação Computacional	18
3.1.1	Limite de Integração Λ	18
3.1.2	Integração Numérica	18
3.2	Funções Disponíveis	19
A	Biblioteca de Funções	34
B	Programa em C	56

Capítulo 1

Introdução

A suavização pelo método do núcleo estimador é um método não paramétrico bastante difundido na estimação da função densidade de probabilidade ou na função de distribuição de probabilidade subjacente a um conjunto de dados observados. Ele é um método aplicado também na estimação da função de intensidade de um processo de Poisson não-homogêneo e de uma função de regressão.

Uma questão crucial na aplicação deste método é a determinação do parâmetro de suavização ou janela h , que controla o grau de suavização dos dados. Se h é muito pequeno, admite-se demasiado ruído amostral e se h é muito grande, perdem-se características da curva através da super-suavização.

A taxa de convergência e a suavidade do estimador dependem desta escolha da largura da janela, tornando-se de extrema relevância estudar estimadores de janela ótima para a obtenção da estimação mais apropriada da função que exprima a probabilidade de ocorrência dos dados. Sabe-se que a largura da janela ótima do núcleo estimador (h_{op}) da função densidade é da ordem de $n^{-1/5}$ e no caso da função de distribuição, é da ordem de $n^{-1/3}$.

A literatura aborda de várias maneiras a escolha da janela ótima h_{op} . Embora, na prática, seja possível escolher o parâmetro de suavização de maneira subjetiva, há uma grande demanda por procedimentos automáticos para seleção da janela. O seletor automático mais estudado é o da função escore de validação cruzada de mínimos quadrados, tendo sido provado que o minimizador da função escore de validação cruzada é uma estimativa con-

sistente da janela ótima, com normalidade assintótica. Entretanto, dos resultados assintóticos dos procedimentos de validação cruzada, verifica-se que a estimativa da janela está sujeita a uma grande variação amostral. Estudos de simulação indicaram que o seletor tende a escolher valores de janela menores, com mais frequência que o predito pelos teoremas assintóticos.

Outra abordagem possível na escolha da janela ótima é através da utilização de método ‘plug-in’ que estima o valor da única quantidade desconhecida na expressão que define o valor ótimo de h , ou seja, a parcela dependente da função que se quer estimar ($\int [f'']^2$, no caso da estimação da função de densidade ou $\int [F'']^2$ no caso da estimação da função de distribuição). Salienta-se que o método ‘plug-in’, quando aplicável, tem a aparente vantagem de, em seu cálculo, não necessitar de uma rotina de otimização. Os estimadores ‘plug-in’ utilizados no presente trabalho baseiam-se em funções características, cujo comportamento foi estudado por Damasceno [8], no caso da estimação da função de densidade e por Bessegato [2], no caso da estimação da função de distribuição. De qualquer maneira, o problema bastante especial de estimação de função de densidade ou de distribuição na presença de dados censurados e da função taxa de falha não são abordados no presente trabalho. O objetivo principal deste trabalho é apresentar e detalhar um conjunto de funções desenvolvidas em \mathbb{R} que são úteis na estimação de funcional de um conjunto isolado de dados observados, com flexibilidade na determinação de cada passo do método.

Capítulo 2

Estimação Funcional

Infelizmente, a força da modelagem paramétrica é também sua fraqueza. Ao fazermos inferência de um modelo específico, é possível obter um ganho muito grande em eficiência, mas somente se o modelo assumido for pelo menos aproximadamente verdadeiro. Se o modelo assumido não estiver correto, as inferências podem ser piores e inúteis, levando a enganos grosseiros na interpretação dos dados. Os métodos de suavização oferecem uma ponte entre não estabelecer nenhuma hipótese na estrutura formal (abordagem puramente não-paramétrica) e estabelecer hipóteses muito fortes (abordagem paramétrica). Ao adotar uma hipótese relativamente fraca de que a verdadeira densidade é suave, permite que os dados contem ao analista qual é seu verdadeiro padrão.

Há duas importantes maneiras para incorporar os métodos de suavização na análise de dados. Ser capaz de extrair mais informação dos dados que seria possível por uma abordagem puramente não paramétrica, desde que a hipótese de suavidade seja razoável, assim como livrar-se de hipóteses paramétricas rígidas, fornecendo, em conseqüência, análises que são ao mesmo tempo flexíveis e robustas. Os métodos de suavização oferecem com eficiência uma maneira de ressaltar importantes estruturas subjacentes aos dados.

2.1 Estimação da Função de Densidade

O mais utilizado estimador da função de intensidade é o histograma. Nesse gráfico a área da barra é equivalente à proporção de observações no intervalo ao qual pertencem. A escolha da amplitude desses intervalos basicamente controla a suavidade do procedimento. Este estimador tem duas

importantes limitações: a dependência do comprimento do intervalo e o fato de o intervalo não constituir uma curva contínua. Essa última limitação incentivou a procura de estimadores contínuos.

Seja uma variável aleatória contínua X com função de densidade f , então, utilizando a idéia de probabilidade freqüentista, podemos estabelecer um estimador \hat{f} da densidade tomando:

$$\hat{f}(x) = \frac{1}{2nh} [\# \text{ de } X_i' \text{ s } \in (x - h; x + h)]$$

escolhendo-se um número h pequeno. Esta função é chamada de estimador natural.

Para expressar melhor o estimador natural, podemos escrevê-lo como:

$$\hat{f}(x) = \frac{1}{nh} \sum_{j=1}^n w \left(\frac{x - X_j}{h} \right)$$

em que w é uma função peso, definida por:

$$w(x) = \begin{cases} 1/2 & \text{se } |x| < 1 \\ 0 & \text{caso contrário} \end{cases}$$

Para superar algumas das limitações deste estimador, em particular o fato de \hat{f} não ser contínua, generaliza-se sua expressão, tomando a função peso w por uma função núcleo k que satisfaz a condição de que $\int_{-\infty}^{\infty} k(x) dx = 1$. Em geral, assume-se que k é uma função de densidade simétrica. Assim, o núcleo estimador com núcleo k é dado por:

$$\hat{f}(x) = \frac{1}{nh} \sum_{j=1}^n k \left(\frac{x - X_j}{h} \right) \quad (2.1)$$

Pode-se verificar que quando h é escolhido muito pequeno, o resultado da estimativa da função de densidade tende a produzir estruturas falsas que apresentam curvas muito irregulares. Já quando h é escolhido grande o resultado da estimativa da função de densidade tende a super-suavizar f .

Da própria definição do núcleo-estimador, seguem algumas propriedades elementares, dentre as quais a mais importante talvez seja a de que \hat{f} herdará todas as propriedades de continuidade e diferenciabilidade do

núcleo k . Assim, se k for uma função de densidade normal, então \hat{f} será uma curva suave tendo derivadas de todas as ordens.

2.1.1 A Janela Ótima e as Propriedades do Estimador

Para avaliar o desempenho do núcleo estimador \hat{f} da função de densidade é necessário escolher uma medida de distância entre a densidade verdadeira f e o estimador \hat{f} . A maioria dos trabalhos utilizam distâncias L^2 , pois, elas permitem uma análise mais simples que se utilizadas distâncias L^1 . Comumente, escolhe-se o Erro Quadrático Integrado (ISE-Integrated Squared Error), dado por:

$$ISE(h) = \int \left\{ \hat{f}_h(x) - f(x) \right\}^2 dx$$

e seu valor esperado, o Erro Quadrático Médio Integrado (MISE-Mean Squared Error), dado por:

$$MISE(h) = E \left[\int \left\{ \hat{f}_h(x) - f(x) \right\}^2 dx \right]$$

Aqui e no futuro, a integral é tomada em toda a reta, se não estiver indicada de outra maneira.

Utilizando-se propriedades elementares de média e variância, pode-se escrever o $MISE$ como:

$$MISE(h) = \int Var \hat{f}(x) dx + \int vicio^2 \hat{f}(x) dx$$

Percebe-se que o vício pode ser reduzido ao custo do aumento da variância e vice-versa, variando h .

Um bem conhecido critério de desempenho é derivado de uma análise assintótica do $MISE(h)$. Assumindo-se que f é pelo menos 2 vezes diferenciável, então, por mudança de variáveis e através da expansão de Taylor de f é fácil mostrar que a variância integrada assintótica de $\hat{f}(x)$, $IV(h) = \int Var \hat{f}(x) dx$, é dada por:

$$IV(h) = \frac{1}{nh} \int k(t)^2 dt + O\left(\frac{1}{n}\right)$$

e o vício quadrático integrado assintótico de $\hat{f}(x)$, $IB(h) = \int Vicio^2 \hat{f}(x) dx$, é dado por:

$$IB(h) = \frac{1}{4}h^4k_2^2 \int f''(x)^2 dx + o(h^4)$$

onde k_2 é a variância da densidade escolhida como núcleo, ou seja $\int t^2k(t)dt$.

O vício na estimação de $f(x)$ não depende diretamente do tamanho da amostra, mas da amplitude da janela h . A influência da escolha de h no núcleo estimador $\hat{f}(x)$ é facilmente verificada através da análise de $IV(h)$ e de $IB(h)$. Pequenos valores de h aumentam a variância assintótica e portanto a estimativa $\hat{f}(x)$ resultante parecerá muito irregular com muitas características espúrias, se verificada graficamente. Por outro lado, valores grandes de h reduzem a variância assintótica de $\hat{f}(x)$ mas aumenta o vício assintótico, talvez através da suavização excessiva de alguma característica subjacente interessante da densidade verdadeira f . Verifica-se ainda que para $MISE(h) \rightarrow 0$ quando $n \rightarrow \infty$ é necessário e suficiente que $h \rightarrow 0$ tal que $nh \rightarrow \infty$, com $n \rightarrow \infty$.

A partir da expressão do ‘MISE’ assintótico, o valor ótimo de h é dado por:

$$h_{op} = k_2^{-2/5} \left[\int k(t)^2 dt \right]^{1/5} \left[\int f''(x)^2 dx \right]^{-1/5} n^{-1/5} \quad (2.2)$$

Podemos tirar algumas conclusões desta expressão. Primeiramente a janela ótima irá convergir a uma taxa muito baixa para zero, quando o tamanho da amostra aumentar. Segundo, visto que o termo $\int (f'')^2$ mede, em um certo sentido, a rapidez das flutuações na densidade f , valores menores de h serão apropriados para as densidades cuja distribuição é mais flutuante.

Dado que f é contínua em x e supondo que o núcleo k seja um função de densidade limitada e que a janela assumida satisfaça que $h \rightarrow 0$ e $nh \rightarrow \infty$ quando $n \rightarrow \infty$; então pode-se provar que:

$$\hat{f}(x) \rightarrow f(x), \text{ em probabilidade, quando } n \rightarrow \infty$$

A condição de convergência da janela descrita acima implica que, enquanto a janela deve buscar valores pequenos quando o tamanho da amostra aumenta, ela não deve convergir para zero tão rapidamente quanto n^{-1} . Isto significa que o número esperado de pontos na amostra que pertencem ao intervalo $x \pm h$ deve tender ao infinito, embora lentamente, quando o tamanho da amostra tende ao infinito.

2.1.2 Estimador ‘Plug-in’

Da expressão 2.2, percebe-se que a janela não é disponível na prática, pois, ela depende da função de densidade desconhecida f . Há vários métodos propostos para estimar a janela ótima a partir de uma amostra aleatória X_1, \dots, X_n .

Uma aproximação comum na seleção automática da janela é obter uma estimativa do $MISE(h)$ e usar o minimizador da função de risco estimada como uma função de h_{op} , ou seja, um estimador geral do h teórico que minimiza o $MISE(h)$ teórico. O Método da Validação Cruzada por Mínimos Quadrados, proposto por Rudemo [12] e Bowman [3] é provavelmente o mais popular e o mais estudado dos métodos de validação cruzada. A função de Validação Cruzada é dada por:

$$CV_n(h) = \int \hat{f}_h(x)^2 dx - 2n^{-1} \sum_{i=1}^n \hat{f}_{h,i}(X_i) \quad (2.3)$$

onde $\hat{f}_{h,i}(x)$ é o núcleo estimador avaliado com a i -ésima observação retirada da amostra.

A idéia que suporta o método é de que a função a ser minimizada depende apenas dos dados e o valor ótimo da janela é estimado pelo valor de h que minimiza esta função. Entretanto, verifica-se que este método sofre muito com a variação amostral, isto é, para diferentes amostras da mesma distribuição, as janelas estimadas apresentam uma grande variabilidade. Uma outra desvantagem do método é que a função a ser minimizada apresenta freqüentemente diversos mínimos, com alguns deles espúrios situados na região de sub-suavização. Por outro lado, algumas vezes ocorre o problema de não existir mínimo.

A abordagem do método ‘plug-in’ original é usar uma janela h_1 para calcular \hat{f}_{h_1} , e tomar esta estimativa para calcular $\int (\hat{f}'')^2$ e utilizá-la em 2.2 para obter h_{op} . Veja em Woodroffe [18].

O método que utilizamos neste trabalho foi proposto por Chiu [7], que aborda o problema de escolha da janela através da função característica e encontra uma expressão equivalente para a quantidade desconhecida $G = \int (f''(x))^2 dx$.

Notamos que $G = \int [f''(x)]^2 dx$ é a única quantidade desconhecida na expressão para o valor de h_{op} (2.2). Esta consideração leva à consideração do método ‘plug-in’, que obtém uma estimativa da janela substituindo G por uma estimativa de G .

A função característica da densidade f é dada por:

$$\varphi(\lambda) = \int e^{i\lambda x} f(x) dx$$

Pela Fórmula da Inversão tem-se que :

$$f(x) = \frac{1}{2\pi} \int e^{-i\lambda x} \varphi(\lambda) d\lambda$$

Logo,

$$\begin{aligned} f'(x) &= \frac{1}{2\pi} \int (-i\lambda) e^{-i\lambda x} \varphi(\lambda) d\lambda, e \\ f''(x) &= \frac{1}{2\pi} \int (\lambda)^2 e^{-i\lambda x} \varphi(\lambda) d\lambda \end{aligned}$$

Da Identidade de Parseval, tem-se que:

$$\int [f(x)]^2 dx = \frac{1}{2\pi} \int |\varphi(\lambda)|^2 d\lambda$$

Logo,

$$\begin{aligned} G &= \int_{-\infty}^{\infty} [f''(x)]^2 dx \\ &= \frac{1}{2\pi} \int_0^{\infty} \lambda^4 |\varphi(\lambda)|^2 d\lambda \end{aligned} \quad (2.4)$$

A função característica amostral é definida por:

$$\hat{\varphi}(\lambda) = \frac{1}{n} \sum_{j=1}^n e^{i\lambda X_j}$$

A partir desta definição, temos que:

$$|\hat{\varphi}(\lambda)|^2 = \left[\frac{\sum_j \cos(\lambda X_j)}{n} \right]^2 + \left[\frac{\sum_j \text{sen}(\lambda X_j)}{n} \right]^2 \quad (2.5)$$

Para alcançar uma regra de seleção da janela melhor, ele propôs modificar a função característica amostral abaixo de alguma frequência de corte. O método oferece uma seleção completamente automática, com baixo esforço computacional em comparação ao método de validação cruzada.

Primeiramente, encontramos Λ que é o primeiro valor de λ tal que $|\hat{\varphi}(\lambda)|^2 < c/n$ para $c > 1$. Damasceno [8] avaliou o método para os valores de

$c = 1, 2, e 3$, verificando que para $c = 3$ a variância do estimador era menor. Salienta-se que essa constante c impacta no resultado mesmo quando f é suficientemente suave.

A proposta de Chiu [7] é estimar G por:

$$\hat{G} = \frac{1}{\pi} \int_0^\Lambda \lambda^4 \left[|\hat{\varphi}(\lambda)|^2 - \frac{1}{n} \right] d\lambda \quad (2.6)$$

onde Λ é o primeiro valor de λ tal que $|\hat{\varphi}(\lambda)|^2 \leq c/n$ para alguma constante $c > 1$. Damasceno [8] indica a adoção de $c = 3$ uma vez que este valor apresentou menor variância não só nas simulações que realizou, mas também naquelas efetuadas por Chiu. Pode-se provar a consistência forte e a normalidade assintótica de \hat{G} .

Usando estes resultados, a partir da expressão 2.2, estabelece-se o seguinte estimador para a janela ótima:

$$\hat{h}_{op} = k_2^{-2/5} \left[\int k(t)^2 dt \right]^{1/5} \left[\hat{G} \right]^{-1/5} n^{-1/5} \quad (2.7)$$

2.2 Estimação da Função de Distribuição

O método do núcleo estimador também tem sido largamente utilizado na estimação da função de distribuição, consolidando-se como uma alternativa às abordagens paramétricas. Por oportuno, salientamos que as técnicas usadas na estimação da função de densidade não são diretamente aplicáveis no caso da estimação da função de distribuição.

Dada uma amostra aleatória X_1, \dots, X_n , de uma variável aleatória contínua X , com função de distribuição F , define-se o estimador de F , avaliado no ponto x , por:

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^n K \left(\frac{x - X_i}{h_n} \right) \quad (2.8)$$

onde K é uma função de distribuição, sendo denominado núcleo e h_n é chamado de parâmetro de suavidade.

Assumiremos que a função de densidade $k = K'$ é limitada, simétrica, continuamente diferenciável, tem suporte compacto e $0 < \int t^2 k(t) dt = k_2 < \infty$. Assumiremos também que $h_n \rightarrow 0$ e $nh_n \rightarrow \infty$, quando $n \rightarrow \infty$. De maneira análoga ao caso da estimação da função de densidade, a taxa de convergência e a suavidade do núcleo estimador dependem da escolha de

uma largura de janela. A partir daqui, para simplificação, escreveremos h em lugar de h_n e, quando não houver indicação dos limites de integração, assume-se que a integral é sobre toda a reta.

2.2.1 A Janela Ótima e as Propriedades do Estimador

A escolha do núcleo K não é muito crucial, mas a escolha do parâmetro de suavidade é um sério problema que tem sido tratado exaustivamente na literatura. Em geral, h é escolhido de maneira que $\hat{F}_n(x)$ seja um ótimo estimador de F , de acordo com alguma medida de desempenho, sendo comum o uso do Erro Quadrático Médio Integrado, (MISE-Mean Integrated Squared Error) que é definido como:

$$MISE(h) = E \int \left\{ \hat{F}_n(x) - F(x) \right\}^2$$

Está disponível há algum tempo uma expressão para a janela que minimiza o $MISE(h)$, verificando-se que este valor ótimo, h_{op} infelizmente depende da função desconhecida F . Precisamos então estimar h_{op} a partir dos dados observados. De Bowman et al [6], obtemos a expressão da janela ótima:

$$h_{op} = \left\{ \frac{\int K(x)[1 - K(x)] dx}{\left[\int z^2 dK(z) \right]^2 \int [F''(x)]^2 dx} \right\}^{1/3} n^{-1/3} \quad (2.9)$$

Percebe-se aqui uma importante diferença com o caso da estimação da densidade, onde o parâmetro de suavidade tem a forma assintótica de $C_5 n^{-1/5}$.

2.2.2 Estimador ‘Plug-in’

A literatura aborda de várias maneiras a escolha da janela ótima h_{op} . Embora, na prática, seja possível escolher o parâmetro de suavização de maneira subjetiva, há uma grande demanda por procedimentos automáticos para seleção da janela.

De maneira similar àquela utilizada em estimação da função de densidade, Bowman et al [6] propuseram um método de validação cruzada para a suavização de funções de distribuição. Neste método, a seleção do parâmetro da janela é baseado em estimação não-viciada do ‘MISE’ (Mean Integrated Squared Error), levando seu procedimento de minimização a uma escolha assintótica da janela ótima. Como no caso da estimação da função de

densidade, verifica-se a partir de resultados assintóticos dos procedimentos de validação cruzada que a estimativa está sujeita a uma grande variação amostral, sendo que em estudos de simulação observou-se que o seletor tende a escolher valores de janelas menores, com mais frequência que o predito pelos Teoremas assintóticos.

Uma abordagem possível na escolha da janela ótima é através da utilização de método ‘plug-in’, que estima o valor da única quantidade desconhecida na expressão do erro quadrático médio integrado assintótico, ou seja, a parcela $H = \int [F'']^2$ a qual depende da função que se quer estimar. Salienta-se que o método ‘plug-in’ tem a aparente vantagem de, em seu cálculo, não necessitar de uma rotina de otimização.

H é a única quantidade desconhecida na expressão de h_{op} (2.9) e para sua estimação utilizamos um estimador análogo àquele proposto por Chiu [7] no caso da estimação da função de densidade. Essa quantidade é estimada usando a função característica empírica da amostra. Em Atuncar, Bessegato e Duczmal [1], verifica-se que H pode ser aproximada por:

$$\hat{H} = \frac{1}{\pi} \int_0^\Lambda \lambda^2 \left[|\hat{\varphi}(\lambda)|^2 - \frac{1}{n} \right] d\lambda \quad (2.10)$$

onde $\Lambda = \min \left\{ \lambda : |\hat{\varphi}(\lambda)|^2 \leq \frac{C}{n} \right\}$, para algum $C > 1$.

Substituímos \hat{H} em lugar de H na expressão (2.9) e obtemos um estimador \hat{h}_{op} , para h_{op} , ou seja:

$$\hat{h}_{op} = \left\{ \frac{\int K(x)[1 - K(x)] dx}{\left[\int z^2 dK(z) \right]^2 \hat{H}} \right\}^{1/3} n^{-1/3} \quad (2.11)$$

A consistência forte de \hat{H} e de \hat{h}_{op} encontram-se provadas em Bessegato [2].

2.3 Estimação da Função de Intensidade

No caso em que se deseja estimar a função de intensidade de um processo não homogêneo, considera-se que os pontos X_1, \dots, X_n são uma realização no intervalo $[0, T]$ de um processo de Poisson não homogêneo

com função de intensidade λ . O núcleo estimador da função de intensidade λ avaliado no ponto t é dado por:

$$\hat{\lambda}_h(t) = \frac{1}{h} \sum_{i=1}^n k\left(\frac{t - X_i}{h}\right) \quad (2.12)$$

A única diferença em relação ao estimador definido em 2.1 é a ausência do fator n^{-1} , já que essa função pode ser interpretada como o número de observações em um determinado intervalo, enquanto a densidade é uma proporção de eventos.

2.3.1 A Escolha da Janela Ótima

Especificamente para a seleção da janela ótima no caso de estimação da função de intensidade, Diggle [9] apresentou o método Bayesiano empírico. Verifica-se entretanto na literatura que, no caso de estimação da função de intensidade, os métodos Bayesiano Empírico e de Validação Cruzada são equivalentes, no sentido em que selecionam a mesma janela. No entanto, como observado por Chiu [7], o seletor de validação cruzada, definido como o minimizador de $CV(h)$ está sujeito a alta variabilidade e a modificação proposta por Chiu apresenta melhor desempenho. Dessa forma a aplicabilidade do método de validação cruzada no caso de estimação da função de intensidade, é natural pensar no método estabilizado de validação cruzada como um melhor método de seleção da janela ótima também esta situação.

Dada a equivalência entre os métodos estabilizado de validação cruzada e ‘plug-in’ modificado, métodos muito difundidos, com implementação computacional conhecida, Travassos [15] verificou seu bom desempenho no contexto da estimação da função de intensidade.

2.4 Problemas de Fronteira

Quando o suporte da função a ser estimada é limitado, o núcleo estimador tem um desempenho insatisfatório nas fronteiras do intervalo. Embora se saiba que o núcleo estimador seja viciado, perto das fronteiras do intervalo o vício do estimador é ainda maior que no seu interior, com tendência à subestimação.

A idéia por trás do método da reflexão é transformar os dados de modo que, na escala transformada, a função a ser estimada tenha suporte ilimitado. Isto é feito refletindo-se os pontos em torno do ponto de fronteira.

Para correção deste problema já foram propostas várias soluções. Devido a sua grande simplicidade de implementação e a sua natureza intuitiva, os métodos da reflexão se destacam dos demais. Travassos [15] estudou estes métodos, adotando a escolha da janela pelo método ‘plug-in’ modificado.

Implementamos rotina para o caso em $X \geq 0$. Neste caso, o núcleo estimador corrigido é dado por:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n \left[k \left(\frac{x - X_i}{h} \right) + k \left(\frac{-x - X_i}{h} \right) \right]. \quad (2.13)$$

A janela h é selecionada através do método ‘plug-in’ modificado, já adotado nas demais estimações funcionais.

Capítulo 3

Funções em R

O objetivo do conjunto de funções apresentado neste trabalho é o de efetuar estimação funcional de um único conjunto de dados. Neste caso, as funcionalidades esperadas são a flexibilidade e a interatividade com o usuário, principalmente pela facilidade na manipulação das funções. Estas são características convenientes às aplicações práticas da metodologia de interesse. Essas funções foram desenvolvidas durante o estudo detalhado do comportamento do método ‘plug-in’ através de simulações intensivas. Nesta ocasião, buscávamos o controle das etapas mais importantes de todo esse processo. Optamos pela utilização do R [11], que é um ambiente de software livre para estatística computacional e produção de gráficos. Neste ambiente há algumas bibliotecas relacionadas com núcleo estimador. Destacamos a biblioteca *sm*, de autoria de Bowman e Azzalini [5], desenvolvida para GNU R e a biblioteca *KernSmooth*, de Wand e Jones [16]. Ambos os pacotes enfatizam a estimação da densidade, dando-se a escolha da janela através de variações do método de validação cruzada.

A biblioteca está disponível no endereço <ftp://ftp.est.ufmg.br/pub/nucleo> ou diretamente com o primeiro autor ¹. Ela pode ser utilizada para a estimação da função de distribuição, da função de densidade de probabilidade ou da função de intensidade de um processo, possibilitando o cálculo das seguintes etapas do procedimento de estimação: determinação do limite de integração Λ , cálculo de \hat{H} (ou \hat{G} , cálculo de h_{op} , cálculo e gráfico de \hat{F}_n). A tabela 3.1 apresenta um resumo das funções disponíveis.

Para a execução de simulações intensivas ou de estudos mais aprofundados da metodologia, recomendamos o uso da rotina em C, disponibi-

¹E-mail: lupercio@est.ufmg.br

Tabela 3.1: Resumo das Funções das Principais Funções da Biblioteca

Comando	Argumentos	Descrição
func.phi2	(vetor.dados,x)	Calcula $ \hat{\varphi}(\lambda) ^2$
alg.lambda	(vetor.dados, cota.sup=10, cota.inf=0)	Calcula Λ
alg.G	(vetor.dados, lambda)	Calcula \hat{G}
alg.H	(vetor.dados, lambda)	Calcula \hat{H}
alg.hop.f	(vetor.dados, \hat{G} ,flag)	Calcula h_{op} densidade
alg.hop	(vetor.dados, \hat{H} ,flag)	Calcula h_{op} distribuição
func.F.norm	(x, h_{op} ,vetor.dados)	Calcula $\hat{F}_n(x)$ (núcleo gaussiano)
plug.smooth	(vetor.dados,rotulo=NULL)	Estimação da densidade
plug.smooth.F	(vetor.dados,rotulo=NULL)	Estimação da distribuição
plug.trans	(vetor.dados,rotulo=NULL)	Estimação da densidade com transformação
plug.trans.F	(vetor.dados,rotulo=NULL)	Estimação da distribuição com transformação
plug.reflex	(vetor.dados,rotulo=NULL)	Estimação da densidade com correção efeito de fronteira
plug.intensidade	(vetor.dados,rotulo=NULL)	Estimação intensidade do processo

lizada no Apêndice B deste relatório. Sua utilização assegurará uma grande capacidade de memória, haja visto a quantidade de amostras e de cálculos necessários ao desenvolvimento deste tipo de trabalho. O programa em C não faz uso de nenhuma biblioteca estabelecida, tendo sido desenvolvidas todas as funções necessárias ao algoritmo de simulação, além de rotinas gráficas para a construção dos histogramas das distribuições envolvidas.

3.1 Implementação Computacional

3.1.1 Limite de Integração Λ

Preocupou-nos inicialmente a precisão na escolha do limite superior de integração Λ . Através de simulações, verificou-se que valores da ordem de até 10^{-3} são suficientes para assegurar a precisão da integração numérica necessária à obtenção do valor de \hat{H} (\hat{G}).

Ressalta-se que, segundo a literatura, para o caso da estimativa de f , a escolha do valor da constante c não é importante, quando a função é suficientemente suave e n é grande. Adotamos o valor sugerido por Chiu para o seletor estabilizado ($c = 3$) e verificamos que a exatidão dos resultados não era significativamente afetada quando se escolhiam os valores de c na faixa sugerida por Chiu, ou seja, entre $-\ln(0,15)$ e $-\ln(0,05)$.

3.1.2 Integração Numérica

Por estar presente na maioria das etapas do processo de estimação, a integração numérica foi uma parte importante do desenvolvimento computacional deste trabalho. O método de Simpson, conforme Mathews [10], mostrou-se adequado às integrações numéricas, dadas suas características de robustez, fornecendo uma precisão razoável aos resultados. Utilizamos este processo na estimação de H e na obtenção dos valores de \hat{F}_n , a partir da estimativa de densidade f_n . Para obter-se precisão na obtenção de \hat{H} , verificamos ser suficiente a utilização de um passo de tamanho 0,1; não havendo uma alteração significativa quando utilizado um passo de 0,01. Entretanto, no processo de validação do algoritmo, verificamos que em algumas situações o passo de tamanho 0,1 poderia influenciar os resultados das estimativas de F obtidas através da $\int f_n$. Assim, optamos pela utilização do passo de tamanho 0,01 em ambas as situações, de maneira a termos garantida a precisão dos valores obtidos. Para o cálculo do 'ISE', por tratar-se da integração de uma diferença ao quadrado, utilizamos a integral de Riemann, com um passo de tamanho 0,01, procedimento que se mostrou estável nas validações efetuadas, obtendo-se uma precisão razoável.

Salientamos, entretanto, que o principal cuidado a ser tomado refere-se ao intervalo de truncamento da integral, já que grande parte dos cálculos e simulações envolvem integrais que possuem pelo menos um de seus limites tendendo ao infinito. Utilizamos assim intervalos de integração grandes o suficiente para garantir a precisão dos resultados e a correção de nossas conclusões.

3.2 Funções Disponíveis

<code>func.phi2</code>	<i>Módulo da Função Característica Empírica</i>
------------------------	---

Descrição

Calcula o quadrado do módulo da função característica empírica, $|\hat{\varphi}(\lambda)|^2$

Sintaxe

```
func.phi2(vetor.dados,lambda)
```

Argumentos

<code>vetor.dados</code>	vetor com os dados observados
<code>lambda</code>	valor da frequência λ em que se deseja calcular o módulo da função característica empírica.

Detalhes

Esta função é intensivamente utilizada nas demais rotinas e deve ser carregada para permitir o funcionamento correto de todas as outras funções. Calcula a expressão 2.5 que é essencial na aplicação da metodologia em questão.

Autor

Lupércio F. Bessegato, 2001.

<code>alg.lambda</code>	<i>Determinação do Limite Superior de Integração</i>
-------------------------	--

Descrição

Pesquisa o limite superior de integração Λ

Sintaxe

```
alg.lambda(vetor.dados,cota.sup=4,cota.inf=0)
```

Argumentos

`vetor.dados` vetor com os dados observados
`cota.superior` máximo valor de Λ imposto na pesquisa
`cota.inferior` limite inferior para a pesquisa de Λ

Detalhes

O uso de cota superior e cota inferior na estimação de Λ busca agilizar a pesquisa. Nos casos em que $\Lambda > 4$ a função não retornará nenhum valor (N/A). A pesquisa deverá ser retomada com valores maiores para `cota.superior`, até ocorrer uma saída numérica desta função.

A função considera internamente o valor de $c = 3$, conforme explicado no corpo deste relatório. No decorrer das simulações em Bessegato (2001)[2] verificou-se que valores de Λ com precisão até a terceira casa decimal eram suficiente para garantir a eficiência das estimativas.

Autor

Lupércio F. Bessegato, 2001.

`alg.G`

Estimativa de G

Descrição

Calcula a estimativa de $G = \int [f''(x)]^2 dx$

Sintaxe

`alg.G(vetor.dados, lambda)`

Argumentos

`vetor.dados` vetor com os dados observados
`lambda` limite superior de integração Λ

Detalhes

G é a única quantidade desconhecida na expressão para o valor de h_{op} 2.2. A rotina calcula \hat{G} (2.6) utilizando o método de Simpson. Considera internamente uma quantidade fixa de passos utilizado pelo método de integração. Bessegato [2] verificou ser esta quantidade adequada para a precisão das simulações então efetuadas. Recomenda-se que quantidade $int = 50$ seja alterada na rotina interna da função, caso haja suspeita acerca da precisão da integração.

Autor

Lupércio F. Bessegato, 2001.

`alg.hop.dens`

Estimativa da Janela Ótima (densidade)

Descrição

Calcula a estimativa da janela ótima do núcleo estimador da densidade, \hat{h}_{op}

Sintaxe

`alg.hop.dens(vetor.dados,G.hat,flag=0)`

Argumentos

<code>vetor.dados</code>	vetor com os dados observados
<code>G.hat</code>	estimativa obtida, \hat{G}
<code>flag</code>	núcleo a ser utilizado na estimação. (flag=0 : núcleo gaussiano, flag=1 : núcleo de Epanechnikov)

Detalhes

Calcula a estimativa da janela ótima, através da expressão 2.7. Caso haja interesse em usar outro tipo de núcleo simétrico, devem ser modificados na expressão, os valores referentes à variância do núcleo adotado e da integral do núcleo ao quadrado, que fazem parte da expressão 2.7.

Autor

Lupércio F. Bessegato, 2001.

`func.f.dens.norm`

Cálculo de $\hat{f}_n(x)$ - núcleo gaussiano

Descrição

Calcula o núcleo estimador da densidade em x , através do núcleo gaussiano

Sintaxe

`func.f.dens.norm(x,hop,vetor.dados)`

Argumentos

<code>x</code>	ponto em que se deseja obter estimativa da função de densidade
<code>hop</code>	estimativa da janela ótima \hat{h}_{op} (núcleo gaussiano)
<code>vetor.dados</code>	dados amostrais

Detalhes

Calcula a estimativa da função de densidade em x ($\hat{f}(x)$), de acordo com a expressão 2.1. O núcleo k é a função de densidade normal padrão.

Autor

Lupércio F. Bessegato, 2001.

`func.f.dens.epnk`

Cálculo de $\hat{f}_n(x)$ - núcleo de Epanechnikov

Descrição

Calcula o núcleo estimador da densidade em x , através do núcleo de Epanechnikov

Sintaxe

`func.f.dens.epnk(x,hop,vetor.dados)`

Argumentos

<code>x</code>	ponto em que se deseja obter estimativa da função de densidade
<code>hop</code>	estimativa da janela ótima \hat{h}_{op} (núcleo de Epanechnikov)
<code>vetor.dados</code>	dados amostrais

Detalhes

Calcula a estimativa da função de densidade em x ($\hat{f}(x)$), de acordo com a expressão 2.1. O núcleo k é o núcleo de Epanechnikov, definido de acordo com Bowman [6].

Autor

Lupércio F. Bessegato, 2001.

`alg.H`

Estimativa de H

Descrição

Calcula a estimativa de $H = \int [F''(x)]^2 dx$

Sintaxe

`alg.H(vetor.dados, lambda)`

Argumentos

<code>vetor.dados</code>	vetor com os dados observados
<code>lambda</code>	limite superior de integração Λ

Detalhes

H é a única quantidade desconhecida na expressão para o valor de h_{op} 2.9. A rotina calcula \hat{H} (2.10) utilizando o método de Simpson. Considera internamente uma quantidade fixa de passos utilizado pelo método de integração. Bessegato [2] verificou ser esta quantidade adequada para a precisão das simulações então efetuadas. Recomenda-se que quantidade `int = 50` seja alterada na rotina interna da função, caso haja suspeita acerca da precisão da integração.

Autor

Lupércio F. Bessegato, 2001.

alg.hop.F

Estimativa da Janela Ótima (distribuição)

Descrição

Calcula a estimativa da janela ótima do núcleo estimador da função de distribuição, \hat{h}_{op}

Sintaxe

alg.hop.F(vetor.dados,H.hat,flag=0)

Argumentos

vetor.dados	vetor com os dados observados
H.hat	estimativa obtida, \hat{H}
flag	núcleo a ser utilizado na estimação. (flag=0 : núcleo gaussiano, flag=1 : núcleo de Epanechnikov)

Detalhes

Calcula a estimativa da janela ótima, através da expressão 2.11. Caso haja interesse em usar outro tipo de núcleo simétrico, devem ser modificados na expressão, os valores referentes à variância do núcleo adotado e da integral $\int K[1 - K]$, que fazem parte da expressão 2.11.

Autor

Lupércio F. Bessegato, 2001.

func.F.norm

Cálculo de $\hat{F}_n(x)$ - núcleo gaussiano

Descrição

Calcula o núcleo estimador da função de distribuição em x , através do núcleo gaussiano

Sintaxe

func.F.norm(x,hop,vetor.dados)

Argumentos

<code>x</code>	ponto em que se deseja obter estimativa da função de distribuição
<code>hop</code>	estimativa da janela ótima \hat{H}_{op} (núcleo gaussiano)
<code>vetor.dados</code>	dados amostrais

Detalhes

Calcula a estimativa da função de distribuição em x ($\hat{F}(x)$), de acordo com a expressão 2.8. O núcleo K é a função de distribuição normal padrão.

Autor

Lupércio F. Bessegato, 2001.

`func.F.epnk`

Cálculo de $\hat{f}_n(x)$ - núcleo de Epanechnikov

Descrição

Calcula o núcleo estimador da função de distribuição em x , através do núcleo de Epanechnikov

Sintaxe

`func.F.epnk(x, hop, vetor.dados)`

Argumentos

<code>x</code>	ponto em que se deseja obter estimativa da função de distribuição
<code>hop</code>	estimativa da janela ótima \hat{h}_{op} (núcleo de Epanechnikov)
<code>vetor.dados</code>	dados amostrais

Detalhes

Calcula a estimativa da função de distribuição em x ($\hat{F}(x)$), de acordo com a expressão 2.8. O núcleo K é o núcleo de Epanechnikov, definido de acordo com Bowman [6].

Autor

Lupércio F. Bessegato, 2001.

<code>alg.percentil</code>	<i>Cálculo de percentil através de núcleo estimador</i>
----------------------------	---

Descrição

Calcula o percentil através do núcleo estimador da função de distribuição em x , utilizando o núcleo gaussiano

Sintaxe

```
alg.percentil(hop, vetor.dados, percentil = 0.95)
```

Argumentos

<code>hop</code>	estimativa da janela ótima \hat{h}_{op} (núcleo gaussiano)
<code>vetor.dados</code>	dados amostrais
<code>percentil</code>	percentil desejado

Detalhes

Calcula a estimativa de percentil através do núcleo gaussiano. Adotado o percentil 0,95 como default.

Autor

Lupércio F. Bessegato, 2002.

<code>plug.smooth</code>	<i>Estimação da Função de Densidade de um Conjunto de Dados</i>
--------------------------	---

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de densidade de um conjunto de dados, fornecendo suas estatísticas descritivas, seu histograma e o gráfico da densidade estimada.

Sintaxe

```
plug.smooth(vetor.dados, rotulo=NULL)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais

Detalhes

Calcula a estimativa da função de densidade através de núcleo estimador gaussiano, com janela escolhida de acordo com o método ‘plug-in’, conforme exposto neste relatório. A função pode ser verificado com o auxílio do conjunto de dados do vetor geysler, referente a dados de tempo de erupção do geysler Old Faithfull, muito utilizado na literatura.

Autor

Lupércio F. Bessegato, 2006.

<code>plug.smooth.F</code>	<i>Estimação da Função de Distribuição de um Conjunto de Dados</i>
----------------------------	--

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de distribuição de um conjunto de dados, fornecendo suas estatísticas descritivas, seu histograma e o gráfico da densidade estimada.

Sintaxe

```
plug.smooth.F(vetor.dados,rotulo=NULL)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais

Detalhes

Calcula a estimativa da função de distribuição através de núcleo estimador gaussiano, com janela escolhida de acordo com o método ‘plug-in’, conforme exposto neste relatório. A função pode ser verificado com o auxílio do conjunto de dados do vetor geysler, referente a dados de tempo de erupção do geysler Old Faithfull, muito utilizado na literatura.

Autor

Lupércio F. Bessegato, 2006.

`plug.trans`

Estimação da Função de Densidade de um Conjunto de Dados com transformação

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de densidade de um conjunto de dados transformado, fornecendo seu desvio-padrão, seu histograma e o gráfico da densidade estimada.

Sintaxe

```
plug.trans(vetor.dados,rotulo=NULL)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais

Detalhes

Calcula a estimativa da função de densidade através de núcleo estimador gaussiano, com janela escolhida de acordo com o método ‘plug-in’, após diminuição de sua variabilidade por divisão dos dados pelo desvio-padrão amostral. A função pode ser verificado com o auxílio do conjunto de dados do vetor `mining.dados`, referentes a dados de acidentes de minas e bastante utilizado na literatura.

Autor

Lupércio F. Bessegato, 2006.

`plug.trans.F`

Estimação da Função de Distribuição de um Conjunto de Dados com transformação

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de distribuição de um conjunto de dados transformado, fornecendo seu desvio-padrão, seu histograma e o gráfico da estimativa da função de distribuição acumulada.

Sintaxe

```
plug.trans.F(vetor.dados,rotulo=NULL)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais

Detalhes

Calcula a estimativa da função de distribuição acumulada através de núcleo estimador gaussiano, com janela escolhida de acordo com o método ‘plug-in’, após diminuição de sua variabilidade por divisão dos dados pelo desvio-padrão amostral. A função pode ser verificado com o auxílio do conjunto de dados do vetor `mining.dados`, referentes a dados de acidentes de minas e bastante utilizado na literatura.

Autor

Lupércio F. Bessegato, 2006.

<code>func.reflex</code>	<i>Cálculo de $\hat{f}_n(x)$, com correção do efeito de fronteira</i>
--------------------------	--

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de densidade de um conjunto de dados, com correção dos efeitos de fronteira, fornecendo suas estatísticas descritivas, seu histograma e o gráfico da densidade estimada.

Sintaxe

```
func.reflex(vetor.dados,rotulo=NULL,fronteira=0)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais
<code>fronteira</code>	limite inferior da variável em estudo

Detalhes

Calcula a estimativa da função de densidade em x , $\hat{f}(x)$ (2.1), adotando a correção de fronteira conforme 2.13. O núcleo k é a função de densidade normal padrão.

Autor

Lupércio F. Bessegato, 2006.

<code>func.intensidade</code>	<i>Cálculo de $\hat{\lambda}(t)$ - núcleo estimador da função de intensidade de um processo</i>
-------------------------------	--

Descrição

Calcula o núcleo estimador da intensidade de um processo não homogêneo no ponto t .

Sintaxe

`func.intensidade(t,hop,vetor.dados)`

Argumentos

<code>t</code>	ponto em que se deseja obter estimativa da função de intensidade
<code>hop</code>	estimativa da janela ótima \hat{h}_{op}
<code>vetor.dados</code>	limite inferior da variável em estudo

Detalhes

Calcula a estimativa da função de intensidade em x , $\hat{f}(x)$ (2.1), adotando a correção de fronteira conforme 2.13. O núcleo k é a função de densidade normal padrão.

Autor

Lupércio F. Bessegato, 2006.

<code>plug.intensidade</code>	<i>Estimação da Função de Intensidade de um Conjunto de Dados</i>
-------------------------------	---

Descrição

Estima e fornece os parâmetros necessários para a estimação da função de intensidade de um conjunto de dados, com correção dos efeitos de fronteira, fornecendo seu desvio-padrão, seu histograma e o gráfico da intensidade estimada.

Sintaxe

```
plug.intensidade(vetor.dados,rotulo=NULL,fronteira=0)
```

Argumentos

<code>vetor.dados</code>	dados amostrais
<code>rotulo</code>	identificação dos dados amostrais

Detalhes

Calcula a estimativa da função de intensidade através de núcleo gaussiano, com janela escolhida de acordo com o método ‘plug-in’. A correção do efeito de fronteira se dá através de 2.13. A função pode ser verificado com o auxílio do conjunto de dados do vetor `mining.dados`, referentes a dados de acidentes de minas e bastante utilizado na literatura. É necessário o cálculo da soma acumulada dos dados para a estimação da função de intensidade.

Autor

Lupércio F. Bessegato, 2006.

Agradecimento

O primeiro autor agradece o apoio financeiro parcial da Fapemig.

Referências Bibliográficas

- [1] ATUNCAR, G. S., BESSEGATO, L. F., AND DUCZMAL, L. H. A consistent estimator for the optimal bandwidth: the distribution function case. Tech. Rep. RTP-06/2003, UFMG/Dep. de Estatística, 2003.
- [2] BESSEGATO, L. F. *Escolha do parâmetro de suavidade na estimativa da função de distribuição*. PhD thesis, UFMG/Departamento de Estatística, Belo Horizonte, Minas Gerais, Brasil, 2001.
- [3] BOWMAN, A. W. An alternative method of cross validation for the smoothing of density estimates. *Biometrika* 71 (1984), 353–360.
- [4] BOWMAN, A. W., AND AZZALINI, A. *Applied smoothing techniques for data analysis: the kernel approach with S-plus Illustrations*. Oxford University Press, New York, 1997.
- [5] BOWMAN, A. W., AND AZZALINI, A. *sm: Smoothing methods for non-parametric regression and density estimation*, 2005. R package version 2.1-0.
- [6] BOWMAN, A. W., HALL, P., AND PRVAN, T. Bandwidth selection for the smoothing of distribution functions. *Biometrika* 85 (1998), 799–808.
- [7] CHIU, S. T. Bandwidth selection for kernel density estimation. *Annals of Statistics* 33 (1991), 1883–1905.
- [8] DAMASCENO, E. C. *Escolha do parâmetro de suavidade em estimação funcional*. PhD thesis, UFMG/Departamento de Estatística, Belo Horizonte, Minas Gerais, Brasil, 2000.
- [9] DIGGLE, P. J. A kernel method for smoothing point process data. *Applied Statistics* 34 (1985), 138–147.

- [10] MATHEWS, J. H. *Numerical methods for mathematics, science and engineering*. Prentice Hall, Englewood Cliffs, 1992.
- [11] R DEVELOPMENT CORE TEAM. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.
- [12] RUDEMO, M. Empirical choice of histograms and kernel density estimators. *Scandinavian Journal of Statistics* 9 (1982), 65–78.
- [13] SILVERMAN, B. W. *Density estimation for statistics and data analysis*. Chapman and Hall, London, 1986.
- [14] SIMONOFF, J. F. *Smoothing methods in statistics*. Springer, New York, 1996.
- [15] TRAVASSOS, A. P. A. *Problemas de fronteira dos núcleos estimadores e suas abordagens*. PhD thesis, UFMG/Departamento de Estatística, Belo Horizonte, Minas Gerais, Brasil, 2003.
- [16] WAND, M. *KernSmooth: Functions for kernel smoothing for Wand & Jones (1995)*, 2005. R package version 2.22-15.
- [17] WAND, M. P., AND JONES, M. C. *Kernel Smoothing*. Chapman and Hall, London, 1995.
- [18] WOODROOFE, M. On choosing a delta sequence. *Ann. Math. Statist.* 19 (1987), 417–442.

Apêndice A

Biblioteca de Funções

```
func.phi2<-function(vetor.data, lambda){
# *** Rotina para Cálculo do Quadrado do Módulo
#     da Função Característica Empírica
#
#   vetor.data  : dados amostrais
#   lambda      : valor de lambda de interesse
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
# *****

  n <- length(vetor.data)
  soma.real <- 0
  soma.imag <- 0
  for(j in 1:n) {
    arg <- lambda * vetor.data[j]
    soma.real <- soma.real + cos(arg)
    soma.imag <- soma.imag + sin(arg)
  }
  phi2 <- ((soma.real)^2 + (soma.imag)^2)/(n^2)
  phi2
}
```

```

alg.lambda<-function(vetor.data, cota.sup = 10, cota.inf = 0){
#
# **** Algoritmo para Pesquisa de Lambda
#
#   vetor.data   : dados amostrais
#   cota.sup     : ma'ximo valor de lambda
#   cota.inf     : minimo valor de lambda
#   n           : tamanho da amostra
#   corte       : ordenada para determinacao de Lambda
#   prec        : precisao do valor de lambda
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# **** Declaracao de Variaveis *****
#   n <- length(vetor.data)
#   corte <- 3/n
#   prec <- 0.001   # Precisao ate'a 3a. casa
# *****
# **** Algoritmo *****
#   vetor.phi2 <- numeric(0)
#   vetor.lambda <- seq(cota.inf, cota.sup, prec)
#   vetor.phi2 <- func.phi2(vetor.data, vetor.lambda)
#   cota.sup <- min(vetor.lambda[vetor.phi2 <= corte])
#   cota.sup
}

alg.H<-function(vetor.dados, lambda, int = 50){
#
# **** Algoritmo para Ca'lculo de H^ - Unica Amostra ****
# ***** Declaracao de Variaveis *****
# Vetor.dados   : dados amostrais
# n             : tamanho da amostra
# int           : qte. de sub-intervalos Integracao de Simpson
# lambda        : Valor da pesquisa de Lambda
# *****

```

```

#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# **** Declaracao de variaveis ****
  n <- length(vetor.dados)  #
#
# ***** Calculo de H^ (Integracao p/ Simpson) *****
#
  vet <- vetor.dados
  h <- lambda/(2 * int)
  soma.par <- 0
  for(k in 1:(int - 1)) {
    x <- 2 * h * k
    func <- (func.phi2(vet, x) - 1/n) * (x^2)
    soma.par <- soma.par + func
  }
  soma.impar <- 0
  for(k in 1:int) {
    x <- h * (2 * k - 1)
    func <- (func.phi2(vet, x) - 1/n) * (x^2)
    soma.impar <- soma.impar + func
  }
# Obs.: func(0)=0
  func.lambda <- (func.phi2(vet, lambda) - 1/n) * (lambda^2)
  soma <- (h * (func.lambda + 2 * soma.par + 4 * soma.impar))/3

#***** Calculo de H^ *****
  aga <- soma/pi
  aga
}

alg.hop.F<-function(vetor.dados, H.hat, flag = 0){
#
# *** Algoritmo para Calculo de h_opt p/ Fn - Amostra Unica ***
# Nucleo Gaussiano e de Epanechnikov
# Vetor.dados : dados amostrais
# Flag : Kernel escolhido - (0) Gaussiano (2) Epanechnikov
# n : numero de elementos de cada amostra

```

```

# int.bow      : integral de Bowman para o nucleo escolhido
# var         : variancia do nucleo escolhido
# cte         : (int.bowman/((variancia do nucleo)^2*n))^1/3
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# *** Declaracao de Variaveis ***
  if(flag == 0) {
# Nucleo Gaussiano
    int.bow <- 0.56418958323
    var <- 1
  }
  else {
# Nucleo Epanechnikov
    int.bow <- 0.257142857
    var <- 0.2
  }
  n <- length(vetor.dados)
  cte <- exp(log(int.bow/(var * var * n))/3)
  H.hat # H^
  h.opt <- cte/exp(log(H.hat)/3)
  h.opt
}

func.F.epnk<-function(x, hop, vetor.dados){
#
# *** Algoritmo para Calculo de Fn^ ***
# Nucleo Epanechnikov (cfe. Bowman)
# vetor.dados: vetor de dados
# hop          : largura da janela ótima
# n            : numero de elementos da amostra
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG

```

```

#
#
# *** Declaracao de Variaveis ***
n <- length(vetor.dados)
soma <- 0
for(i in 1:n) {
  argh <- ((x - vetor.dados[i])/hop)
  if(abs(argh) <= 1) {
    func <- argh * (3/4 - 1/4 * argh * argh) + 1/2
  }
  else {
    if(argh <= (-1)) {
      func <- 0
    }
    else {
      func <- 1
    }
  }
  soma <- soma + func
}
Fn <- soma/n
Fn
}

func.F.norm<-function(x, hop, vetor.dados){
#
# OBS.: Verificar insercao do vetor de dados na atribuicao da funcao
# *** Algoritmo para Calculo de Fn^ ***
# Nucleo Normal
# vetor.dados: vetor de dados
# lin          : linha da matriz de dados
# MAT.hop      : matriz com h_opt
# hop          : largura da janela ótima
# n            : numero de elementos da amostra
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#

```

```

#
# *** Declaracao de Variaveis ***
  n <- length(vetor.dados)
  soma <- 0
  for(i in 1:n) {
    argh <- ((x - vetor.dados[i])/hop)
    soma <- soma + pnorm(arch, 0, 1)
  }
  Fn <- soma/n
  Fn
}

alg.G<-function(vetor.dados, lambda, int = 50){
#
# **** Algoritmo para Ca'lculo de G^ - Unica Amostra ****
# ***** Declaracao de Variaveis *****
# Vetor.dados   : dados amostrais
# n              : tamanho da amostra
# int            : qte. de sub-intervalos Integracao de Simpson
# lambda        : Valor da pesquisa de Lambda
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# **** Declaracao de variaveis *****
  n <- length(vetor.dados)

#
# ***** Calculo de G^ (Integracao p/ Simpson) *****
#
  vet <- vetor.dados
  h <- lambda/(2 * int)
  soma.par <- 0
  for(k in 1:(int - 1)) {
    x <- 2 * h * k
    func <- (func.phi2(vet, x) - 1/n) * (x^4)
    soma.par <- soma.par + func
  }
}

```



```

soma.impar <- 0
for(k in 1:int) {
  x <- h * (2 * k - 1)
  func <- (func.phi2(vet, x) - 1/n) * (x^4)
  soma.impar <- soma.impar + func
}
# Obs.: func(0)=0
func.lambda <- (func.phi2(vet, lambda) - 1/n) * (lambda^4)
soma <- (h * (func.lambda + 2 * soma.par + 4 * soma.impar))/3

##### Calculo de G^ #####
ge <- soma/pi
ge
}

alg.hop.dens<-function(vetor.dados, G.hat, flag = 1){
#
# *** Algoritmo para Calculo de h_opt p/ fn - Amostra Unica ***
# Nucleo Normal e de Epanechnikov
# Vetor.dados : dados amostrais
# Flag : Kernel escolhido - (0) Gaussiano (2) Epanechnikov
# n : numero de elementos da amostra
# R.K : integral do nucleo ao quadrado
# var : variancia do nucleo escolhido
# cte : (R.K/((variancia do nucleo)^2*n))^1/5
#
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# *** Declaracao de Variaveis ***
  if(flag == 0) {
# Nucleo Normal
    R.K <- 0.282093392
    var <- 1
  }
  else {
# Nucleo Epanechnikov

```

```

        R.K <- 0.6
        var <- 0.2
    }
    n <- length(vetor.dados)
    cte <- exp(log(R.K/(var * var * n))/5)
    h.opt <- cte/exp(log(G.hat)/5)
    h.opt
}

func.f.dens.epnk<-function(x, hop, vetor.dados){
#
# *** Algoritmo para Calculo de fn^ ***
# Nucleo Epanechnikov (cfe. Bowman)
# vetor.dados: vetor de dados
# hop          : largura da janela ótima
# n            : numero de elementos da amostra
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# *** Declaracao de Variaveis ***
    n <- length(vetor.dados)
    soma <- 0
    for(i in 1:n) {
        argh <- ((x - vetor.dados[i])/hop)
        if(abs(arch) <= 1) {
            func <- 3/4 * (1 - arch * arch)
        }
        else {
            func <- 0
        }
        soma <- soma + func
    }
    fn <- soma/(n * hop)
    fn
}

func.f.dens.norm<-function(x, hop, vetor.dados){

```

```

#
# *** Algoritmo para Calculo de fn^ (densidade) ***
# Nucleo Normal
# vetor.dados: vetor de dados
# hop          : largura da janela ótima
# n            : numero de elementos da amostra
# *****
#
# Autor:
# (C) 2001, Lupericio F. Bessegato, UFMG
#
#
# *** Declaracao de Variaveis ***
  n <- length(vetor.dados)
  soma <- 0
  for(i in 1:n) {
    argh <- ((x - vetor.dados[i])/hop)
    soma <- soma + dnorm(arch, 0, 1)
  }
  fn <- soma/(n * hop)
  fn
}

alg.percentil<-function(hop, vetor.dados, percentil = 0.95)
{
# ***** Calculo de Percentil *****
# vetor.dados: vetor.dados
# hop          : janela otima
# *****
#
# Autor:
# (C) 2002, Lupericio F. Bessegato, UFMG
#
#
# ***** Declaracao de Variaveis *****
  passo <- 0.001
  a.min <- min(vetor.dados)
  b.max <- max(vetor.dados)
  sig.abcs <- seq(a.min, b.max, by = passo)
  sig.ord <- func.F.norm(sig.abcs, hop, vetor.dados)
}

```

```

# *
sig.sup <- min(sig.abcs[sig.ord >= percentil])
sig.sup
}

plug.smooth<-function(vetor.dados,rotulo=NULL){
#
# Estimativa de Densidade de Conjunto de Dados
# Nucleo Estimador Gaussiano
# Escolha da Janela por Método Plug-in
#
# Declaracao de Variaveis
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
#
#####
#
#
#####

#
# Analise Descritiva dos Dados
#

dpad<- sd(vetor.dados)

cat("\n          RESUMO DOS DADOS\n\n")

print(summary(vetor.dados))
cat("  St. deviation: ",round(dpad,3))

# Pesquisa de Lambda (Limite Superior de Integracao
Lambda=alg.lambda(vetor.dados)
Lambda

```

```

# Estimativa de G
#

Ge=alg.G(vetor.dados,Lambda)
Ge

# Estimativa da Janela - Nucleo Gaussiano
#

flag=0
janela=alg.hop.dens(vetor.dados,Ge,flag)
janela

#
# Estimativas Efetuadas
#

linhas<-c("Lambda","G","Janela")
valores<-c(Lambda, Ge, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n")
print(tabela,width=10)

#
# Graficos
#

pontos<-100

maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

passo<-(maximo-minimo)/pontos
borda<-10*passo

x.abcs<-seq(minimo-borda,maximo+borda,passo)

```

```

#amostra<-"trying"
eixo.x<-paste("",rotulo)
plot(x.abcs,func.f.dens.norm(x.abcs,janela,vetor.dados),type="l",
      ylab="densidade",xlab=eixo.x)
title(main="Density Function Estimation")
#,sub=rotulo)
#windows(hist(vetor.dados))
win.graph()
hist(vetor.dados)
}

plug.smooth.F<-function(vetor.dados,rotulo=NULL){
#
# Estimativa Distribuicao de Conjunto de Dados
# Nucleo Estimador Gaussiano
# Escolha da Janela por Método Plug-in
#
# Declaracao de Variaveis
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
#
#####
#
#
#####

#
# Analise Descritiva dos Dados
#

dpad<- sd(vetor.dados)

cat("\n          RESUMO DOS DADOS\n\n")

print(summary(vetor.dados))
cat("  St. deviation: ",round(dpad,3))

```

```

# Pesquisa de Lambda (Limite Superior de Integracao
Lambda=alg.lambda(vetor.dados)
Lambda

# Estimativa de H
#

aga=alg.H(vetor.dados,Lambda)
aga

# Estimativa da Janela - Nucleo Gaussiano
#

flag=0
janela=alg.hop.F(vetor.dados,aga,flag)
janela

#
# Estimativas Efetuadas
#

linhas<-c("Lambda","H","Janela")
valores<-c(Lambda, aga, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n")
print(tabela,width=10)

#
# Graficos
#

pontos<-100

maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

passo<-(maximo-minimo)/pontos

```

```

borda<-10*passo

x.abcs<-seq(minimo-borda,maximo+borda,passo)

eixo.x<-paste("",rotulo)
plot(x.abcs,func.F.norm(x.abcs,janela,vetor.dados),type="l",
      ylab="Cumulative probability",xlab=eixo.x)
title(main="Distribution Function Estimation")
#,sub=rotulo)
#windows(hist(vetor.dados))
win.graph()
hist(vetor.dados)
}

geyser<-c(4.4,3.9,4,4,3.5,4.1,2.3,4.7,1.7,4.9,1.7,4.6,3.4,4.3,1.7,3.9,3.7,3.1,4,1.
1.9,4.6,2,4.5,3.9,4.3,2.3,3.8,1.9,4.6,1.8,4.7,1.8,4.6,1.9,3.5,4,3.7,3.7,4.3,3.6,3.
4.5,4.1,3.7,3.8,3.4,4,2.3,4.4,4.1,4.3,3.3,2,4.3,2.9,4.6,1.9,3.6,3.7,3.7,1.8,4.6,3.
4.6,1.7,4,1.8,4.4,1.9,4.6,2.9,3.5,2,4.3,1.8,4.1,1.8,4.7,4.2,3.9,4.3,1.8,4.5,2,4.2,
4,4.1,2.7,4.6,1.9,4.5,2,4.8,4.1,4.1,4.2,4.5,1.9,4.7,2,4.7,2.5,4.3,4.4,4.4,4.3,4.6,
4,4,4.4,4.1,4.3,4,3.9,3.2,4.5,2.2,4.7,4.6,2.2,4.8,4.3,3.8,4,4.1,1.8,4.4,4,2.2,5.1,
3.8,4.3,4.4,2.2,4.8,1.9,4.7,1.8,4.8,2,4.4,2.5,4.3,4.4,1.9,4.7,4.3,2.2,4.7,2.3,4.6,
4.6,3.3,4.2,2.6,4.6,3.7,1.8,4.7,4.5,4.5,4.8,2,4.8,1.9,4.7,2,5.1,4.3,4.8,3,2.1,4.6,
4.3,2.1,4.7,4.5,1.7,4.2,4.3,1.7,4.4,4.2,2.2,4.7,4,1.8,4.7,1.8,4.5,2.1,4.2,2.1,5.2,

plug.trans<-function(vetor.dados,rotulo=NULL){
#
# Estimativa de Densidade de Conjunto de Dados c/ Transformacao
# Para conjunto de dados com variancia muito afastada de 1
# Nucleo Estimador Gaussiano
# Escolha da Janela por Método Plug-in
#
# Declaracao de Variaveis
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
#
# *****

```



```

#
# Calculo do Desvio Padrao
#

desvio<-sd(vetor.dados)
cat("\n Desvio-Padrao: ",desvio,"\n")

dados.trans<-vetor.dados/desvio
cota=10
# Pesquisa de Lambda (Limite Superior de Integracao
Lambda=alg.lambda(dados.trans,cota)

# Estimativa de G
#

Ge=alg.G(dados.trans,Lambda)

# Estimativa da Janela - Nucleo Gaussiano
#

flag=0
janela=alg.hop.dens(dados.trans,Ge,flag)

#
# Estimativas Efetuadas
#

linhas<-c("Lambda", "G", "Janela")
valores<-c(Lambda, Ge, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n\n")
print(tabela,width=10)

#
# Graficos
#

```

```

pontos<-100
maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

passo<-(maximo-minimo)/pontos
borda<-10*passo

x.abcs<-seq(minimo-borda,maximo+borda,passo)
eixo.x<-paste("",rotulo)

plot(x.abcs,func.f.dens.norm(x.abcs/desvio,janela,dados.trans)*desvio,type="l",
      ylab="densidade",xlab=eixo.x)
title(main="Density Function Estimation")

#,sub=rotulo)
win.graph()
hist(vetor.dados)
}

plug.trans.F<-function(vetor.dados,rotulo=NULL){
#
# Estimativa Distribuicao de Conjunto de Dados c/ Transformacao
# (Conjunto de dados com variancia muito afastada de 1)
# Nucleo Estimador Gaussiano
# Escolha da Janela por Método Plug-in
#
# Declaracao de Variaveis
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
#
# *****

#
# Calculo do Desvio Padrao
#

```

```

desvio<-sd(vetor.dados)
cat("\n Desvio-Padrao: ",desvio,"\n")

dados.trans<-vetor.dados/desvio
cota=10
# Pesquisa de Lambda (Limite Superior de Integracao
Lambda=alg.lambda(dados.trans,cota)

# Estimativa de G
#

aga=alg.H(dados.trans,Lambda)

# Estimativa da Janela - Nucleo Gaussiano
#

flag=0
janela=alg.hop.F(dados.trans,aga,flag)

#
# Estimativas Efetuadas
#

linhas<-c("Lambda", "H", "Janela")
valores<-c(Lambda, aga, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n\n")
print(tabela,width=10)

#
#Graficos
#

pontos<-100
maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

```

```

passo<-(maximo-minimo)/pontos
borda<-10*passo

x.abcs<-seq(minimo-borda,maximo+borda,passo)
eixo.x<-paste("",rotulo)

plot(x.abcs,func.F.norm(x.abcs/desvio,janela,dados.trans),type="l",
      ylab="Cumulative probability",xlab=eixo.x)
title(main="Distribution Function Estimation")

#,sub=rotulo)
win.graph()
hist(vetor.dados)
}

func.intensidade<-function(x, hop, vetor.dados){
#
# *** Algoritmo para Calculo de lambda^ *****
# *** Estimativa da intensidade no ponto x ***
# Nucleo Normal
# vetor.dados: vetor de dados
# hop          : largura da janela ótima
# n            : numero de elementos da amostra
# *** Declaracao de Variaveis ***
  n <- length(vetor.dados)
  soma <- 0
  for(i in 1:n) {
    argh <- ((x - vetor.dados[i])/hop)
    soma <- soma + dnorm(argh, 0, 1)
  }
  fn <- soma/(hop)
  fn
}

func.reflex<-function(vetor.dados,rotulo=NULL,fronteira=0)
{
# ***** Estimação Função de Densidade *****
# ***** Dados com Problema de Fronteira *****

```

```

# ***** Metodo da Reflexão *****
# ***** Escolha da Janela por Método Plug-in *****
#
# ***** Declaracao de Variaveis *****
#
# vetor.dados : dados amostrais
# rotulo      : Nome da variável em estudo
# fronteira   : Limite inferior da variável em estudo
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
# *****

# Analise Descritiva dos Dados
#

cat("\n          RESUMO DOS DADOS\n\n")

print(summary(vetor.dados))

cota<-10
#
# Pesquisa de Lambda (Limite Superior de Integracao
#
Lambda<-alg.lambda(vetor.y,cota) # Estimacao de Lambda

#
# Estimativa de G
#
Ge<-alg.G(vetor.dados,Lambda)      # Estimacao de G^
#
# Estimativa da Janela - Nucleo Gaussiano
#
flag<- 0                          # Nucleo gaussiano
janela<-alg.hop.dens(vetor.dados,Ge,flag)

#
# Estimativas Efetuadas

```

```

#

linhas<-c("Lambda","G","Janela")
valores<-c(Lambda, Ge, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n")
print(tabela,width=10)
#
# Graficos
#

pontos<-100
maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

passo<-(maximo-minimo)/pontos
borda<-100*passo

x.abcs<-seq(minimo-fronteira,maximo+borda,passo)

ene<-length(x.abcs)
vetor.densidade<-numeric(0)
for(i in 1:ene){ # Cria o vetor
  efe.x<-func.f.dens.norm(x.abcs[i],janela,vetor.dados)
    + func.f.dens.norm(-x.abcs[i],janela,vetor.dados)
  vetor.densidade[i]<-efe.x
}

#
# Grafico
#

eixo.x<-paste("",rotulo)
plot(x.abcs,vetor.densidade,type="l",ylab="densidade",xlab=eixo.x)
title(main="Density Function Estimation")
win.graph()
hist(vetor.dados)
}

```

```

plug.intensidade<- function(vetor.dados,rotulo=NULL,fronteira=0)
{
# ***** Estimação Função de Intensidade *****
# ***** Dados com Problema de Fronteira *****
# ***** Metodo da Reflexão *****
# ***** Escolha da Janela por Método Plug-in *****
#
# ***** Declaracao de Variaveis *****
#
# vetor.dados : dados amostrais
# rotulo      : Variável em estudo
# fronteira   : Fronteira da variável em estudo
#
#
# Autor:
# (C) 2006, Lupericio F. Bessegato, UFMG
#
# *****

    cota<-10
#
# Pesquisa de Lambda (Limite Superior de Integracao
#
    Lambda<-alg.lambda(vetor.dados,cota) # Estimacao de Lambda

#
# Estimativa de G
#
    Ge<-alg.G(vetor.dados,Lambda)      # Estimacao de G^
#
# Estimativa da Janela - Nucleo Gaussiano
#
    flag<- 0                          # Nucleo gaussiano
    janela<-alg.hop.dens(vetor.dados,Ge,flag)

#
# Estimativas Efetuadas
#

```

```

linhas<-c("Lambda","G","Janela")
valores<-c(Lambda, Ge, janela)
tabela<-matrix(valores,ncol=1,byrow=T)
dimnames(tabela)<-list(linhas,"")

cat("\n          ESTIMATIVAS \n")
print(tabela,width=10)
#
# Graficos
#

pontos<-100
maximo<-max(vetor.dados)
minimo<-min(vetor.dados)

passo<-(maximo-minimo)/pontos
borda<-100*passo

x.abcs<-seq(fronteira,maximo+borda,passo)

ene<-length(x.abcs)
vetor.intensidade<-numeric(0)
for(i in 1:ene){ # Cria o vetor
  lambda.x<-func.intensidade(x.abcs[i],janela,vetor.dados)
    + func.intensidade(-x.abcs[i],janela,vetor.dados)
  vetor.intensidade[i]<-lambda.x
}
eixo.x<-paste("",rotulo)
plot(x.abcs,vetor.intensidade,type="l",ylab="intensidade",xlab=eixo.x)
title(main="Intensity Function Estimation")
}

mining.dados<-c(378,36,15,31,215,11,137,4,15,72,96,124,50,120,203,176,
55,93,59,315,59,61,1,13,189,345,20,81,286,114,108,188,233,28,22,61,78,
99,326,275,54,217,113,32,23,151,361,312,354,58,275,78,17,1205,644,467,
871,48,123,457,498,49,131,182,255,195,224,566,390,72,228,271,208,517,
1613,54,326,1312,348,745,217,120,275,20,66,291,4,369,338,336,19,329,
330,312,171,145,75,364,37,19,156,47,129,1630,29,217,7,18,1357)

```


Apêndice B

Programa em C

```
/* determinacao da janela otima para o estimador de distribuicoes H */
/* atraves de integracao por Simpson baseadas em amostras normais */

#include<stdlib.h>
#include<math.h>
#include<stdio.h>
#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define MASK 123459876

int xjmax;
double pi=3.14159265;
double cgauss;
long idum;
char aqui[30];
int normal_impar,nrep,al;
double normal_y,amostran[1000];
double coef_simpson(int i);
double pow(double x, double y);
double sqr(double x);
double normal2(double media, double desv);
double erfff(float x);
double gauss(double x);
```

```

double teste_ad();
void linha(int x0, int y0, int x1, int y1);
void linha1(int x0, int y0, int x1, int y1);
void linha2(int x0, int y0, int x1, int y1);
void linha3(int x0, int y0, int x1, int y1);
void linha4(int x0, int y0, int x1, int y1);
void putp(int x0, int y0, int cor);
void putp1(int x0, int y0, int cor);
void putp2(int x0, int y0, int cor);
void putp3(int x0, int y0, int cor);
void putp4(int x0, int y0, int cor);
float alea(long idum);
domimag()
{
    int i,iii,iiii,j,i1,i2,n,cor,xasp,yasp,a1,a2,b1,b2,
        jm,mx,my,car,band,flagg,ilamb,jj,flag_ker,
        h_opt_i,g_opt_i,ise_i,ise_g_i,xj,ise_emp_i;
    double x1,x2,y1,y2,z1,z2,r1,r2,k1,k2,k3,k4,u1,u2,v1,v2,rr,
        norma,x1min,x2min,x1max,x2max,esc,h,d,asp,aux,r_k,sumfg,Fng,
        stud_emp,stud_g,sump1,sump0,sumpproc,proc0,sumpproc0;

    double c,hc,gc,h_opt,g_opt,pi=3.14159265;
    double x,y,z,ii,delta,ise,ise_g,sump,argx,
        soma,soman,constante,lamb,phi2,soma_real,soma_imag;
    long fd[1000], /* dist. acumulada de f */
        fdp[1000], /* f.d.p. f da v.a. X */
        fdn[1000]; /* fdp normal */
    int
        vetor_hc[1000], /* histograma de valores gerados-H chapeu */
        vetor_h_opt[1000], /* histograma de valores gerados-H optimo */
        vetor_ise[1000], /* histograma de valores gerados-ISE para H */
        vetor_gc[1000], /* histograma de valores gerados-G chapeu */
        vetor_g_opt[1000], /* histograma de valores gerados-G optimo */
        vetor_ise_g[1000], /* histograma de valores gerados-ISE para G */
        vetor_ise_emp[1000]; /* histograma de valores gerados-ISE empirica */
    double vetor_phi2[1000],vetor_lambda[1000];
    long p[4],q[4],tamv,xi,yi,zi,bloco,nn,iphi2,kk;
    double xmin,ymin,zmin,xmax,ymax,zmax,dx,dy,dz,
        nexp,nrep2,nrep4,theta,quiq,cgraf,

```

```

        delta_lambda,corte;
double int_bow, var_ker, cte_ker, inv3, inv5;
double cte_ker_g, g_c, Lambda ;
double m_lambda, s_lambda, d_lambda, s2_lambda;
double m_hc, s_hc, d_hc, s2_hc;
double m_hopt, s_hopt, d_hopt, s2_hopt;
double m_ise, s_ise, d_ise, s2_ise;
double m_gc, s_gc, d_gc, s2_gc;
double m_gopt, s_gopt, d_gopt, s2_gopt;
double m_ise_g, s_ise_g, d_ise_g, s2_ise_g;
double m_pc, s_pc, d_pc, s2_pc;
double m_emp, s_emp, d_emp, s2_emp;
double pv_emp;
double pv_g;
double ise_emp;
double hc0,gc0,h_opt0,g_opt0;
double mse_h,mse_g,mse_hopt,mse_gopt;
double Di_emp,m_Di_emp, d_Di_emp, s2_Di_emp,s_Di_emp;
double Di_g,m_Di_g, d_Di_g, s2_Di_g,s_Di_g;
double m_proc, s_proc, d_proc, s2_proc,mse_proc;
double desvio_amostra,media_amostra2,media_amostra;
double sumpproc0s, sumpproc0s2, m_proc0, d_proc0;
FILE *anormal, *s, *vetor;

setcolor(1);
cor=1;
inv3=1.0/3.0;
inv5=1.0/5.0;
cgauss=sqrt(2.0 * pi);
nexp=1000; /* numero de experimentos (numero de amostras) */
nrep=500; /* numero de repeticoes (tamanho de cada amostra) */
hc0=0.25/sqrt(pi);
gc0=0.2115711;
h_opt0=1.589573060*pow(nrep,-inv3);
g_opt0=1.05922384105*pow(nrep,-inv5);
proc0=0.001;

flag_ker=0; /* 0:Nucleo Normal 1:Nucleo de Epanechnikov */
idum=1; /* semente aleatoria */
anormal=fopen("anormal.dat","w");

```

```

s=fopen("saida.dat","w");
vetor=fopen("vetor.dat","w");
normal_impar=0;
flagg=0; /* 1:aparecem os graficos intermediarios */
tamv=640;
nn=3*nrep/5;
theta=0.4;
cgraf=10;
bloco=32;
nrep4=nrep/4.0;
nrep2=nrep*2.0;
constante=nrep*(1.56/(sqrt(2.0*3.14159265)))/50.0;
if(flag_ker==0){/*normal*/
    int_bow=0.56418958323;
    var_ker=1.0;
    r_k=0.282093392;
}
else if(flag_ker==1){/* Epanechnikov */
    int_bow=0.257142857;
    var_ker=0.2;
    r_k=0.6;
}
cte_ker=int_bow/sqr(var_ker);
cte_ker_g=r_k/sqr(var_ker);
for(i=0;i<1000;i++){vetor_lambda[i]=0;vetor_hc[i]=0;}
for(i=0;i<1000;i++){vetor_h_opt[i]=0;}
for(i=0;i<1000;i++){vetor_ise[i]=0;}
s_lambda=0.0;
s2_lambda=0.0;
s_hc=0;
s2_hc=0;
s_hopt=0;
s2_hopt=0;
s_ise=0;
s2_ise=0;
s_gc=0;
s2_gc=0;
s_gopt=0;
s2_gopt=0;
s_ise_g=0;

```

```

s2_ise_g=0;
s_emp=0;
s2_emp=0;
mse_h=0;
mse_g=0;
mse_hopt=0;
mse_gopt=0;
mse_proc=0;
s_proc=0;
s2_proc=0;
s_Di_emp=0;
s2_Di_emp=0;
s_Di_g=0;
s2_Di_g=0;
sumpproc0=0;
sumpproc0s=0;
sumpproc0s2=0;
for(iiii=0;iiii<nexp;iiii++){
    for(i=0;i<1000;i++){fdp[i]=0;fd[i]=0;}
    if(flagg){
        {setcolor(6);linha(0,470,639,470);}
        {setcolor(1);linha(0,110,639,110);}
        {setcolor(3);linha(319,471,319,479);}
        for(i=0;i<50;i++){putp((int)(4*i)-1,471,3);}
        for(i=0;i<50;i++){putp((int)(4*i)-1,472,3);}
        for(i=0;i<50;i++){putp((int)(4*i)-1,473,3);}
    }
    for(i=0;i<4;i++){p[i]=0;}
    media_amostra=0;
    media_amostra2=0;
    for(jj=0;jj<nrep;jj++){/*geracao amostra c/ nrep valores normais*/
        x=normal2(0,1);
        amostran[jj]=x;
        media_amostra+=x;
        media_amostra2+=sqr(x);
        fprintf(anormal,"%g\n",x);
        quiq=(x+5)/cgraf;
        xi=(int)(tamv*quiq+0.5); if((xi<1000)&&(xi>=0))fdp[xi]++;
        if(flagg)putp(xi,(int)((470-(fdp[xi])/1.0)),2);
    }
}

```

```

desvio_amostra = sqrt( (media_amostra2 - sqr(media_amostra) /
nrep ) / (nrep-1.0) );
media_amostra/=nrep;
for(i=0;i<nrep;i++) /* bolha*/
    for(j=i+1;j<nrep;j++)
        if(amostran[i]>amostran[j]){
            aux=amostran[i];
            amostran[i]=amostran[j];
            amostran[j]=aux;
        }
fprintf(anormal,"\n");
if(flagg){
    soma=0; /* display na tela da amostra normal */
    soman=0;
    for(i=0;i<1000;i++){
        x=(i/64.0-5.0);
        soman+=constante*exp(-x*x/2);
        soma+=fdp[i];
        fd[i]=soma;
        fdn[i]=soman;
    }
    if(flagg){
        putp(i,(int)((470-360*(fd[i])/nrep)),4);
        putp(i,(int)((470-360*(fdn[i])/nrep2)),1);
    }
}
i=-1;
do{
    soma=0;
    j=0;
    do{
        i++;
        soma+=fdp[i];
        j++;
    }while(j<bloco);
    if(flagg){
        setcolor(3);
        linha(i-bloco,(int)(470-360*soma/nn),i,(int)(470-
360*soma/nn));
        linha(i-bloco,470,i-bloco,(int)(470-360*soma/nn));
        linha(i,470,i,(int)(470-360*soma/nn));
    }
}

```

```

}
}while(i<tamv);
clear();
}
/* calcular phi2(lambda) para a amostra normal acima */

if(flagg){
    {setcolor(6);linha(0,470,639,470);}
    {setcolor(1);linha(0,110,639,110);}
}
lamb=0.000;
iphi2=0;
delta_lambda=0.02;
do{
    soma_real=0.0;
    soma_imag=0.0;
    for(iii=0;iii<nrep;iii++){
        soma_real+=cos(lamb*amostran[iii]);
        soma_imag+=sin(lamb*amostran[iii]);
    }
    phi2=soma_real*soma_real+soma_imag*soma_imag;
    phi2=phi2/(nrep*nrep);
    vetor_phi2[iphi2]=phi2;
    iphi2++;
    if(flagg)
        putp((int)((lamb)*100),(int)(470-360*(phi2)),2);
    lamb+=delta_lambda;
}while(lamb<=6.4); /* fim do calculo de phi em [0,6.4] */
corte=3.0/nrep;
iphi2=0;
while(vetor_phi2[iphi2]>corte)iphi2++;
hc=0.0;
for(i=0;i<=iphi2;i++){
    lamb=i*delta_lambda;
    hc+=lamb*lamb*(vetor_phi2[i]-1.0/nrep);
}
Lambda=iphi2*delta_lambda;
hc*=delta_lambda/pi;
fprintf(vetor,"%g ",hc);
if((int)(hc*500)<1000)vetor_hc[(int)(hc*500)]++;

```

```

if(flagg)
    putp((int)(hc*500/3),
          (int)(270-1*(vetor_hc[(int)(hc*500)])),3);
vetor_lambda[iphi2]++;
if(flagg)
    putp((int)((iphi2)*delta_lambda*100),
          (int)(470-1*(vetor_lambda[iphi2])),4);
h_opt=cte_ker/(hc*nrep);
h_opt=pow(h_opt,inv3);
fprintf(vetor,"%g\n",h_opt);
h_opt_i=(int)(h_opt*100+0.5);
if(h_opt*100<1000)vetor_h_opt[h_opt_i]++;
printf("%g ",h_opt);
if(flagg==0)
    putp(h_opt_i, vetor_h_opt[h_opt_i], 5);
delta=0.01;
ise=0.0;
x=-5.0;
while(x<=5.0){
    sump=0.0;
    for(i=0;i<nrep;i++){ /* nucleo estimador em x */
        argx=(x-amostran[i])/h_opt;
        sump+=erfff(argx);
    }
    sump = sump/nrep - erfff(x);
    ise+= sqr(sump);
    x+=delta;
}
ise*=delta;
printf("%g\n",ise);
ise_i=(int)(ise*10000+0.5);
if(ise_i<1000)vetor_ise[ise_i]++;
if(flagg==0)
    putp(ise_i, 200 + vetor_ise[ise_i], 6);

/* process capability (Polanski) */
sump1=0.0;
x=3.291; /* integral de -3.291 a +3.291 e' 0.999 */
for(i=0;i<nrep;i++){

```



```

        argx=(x-amostran[i])/h_opt;
    sump1+=erfff(argx);
}
    sump1 = sump1/nrep;
sump0=0.0;
x=-3.291;
    for(i=0;i<nrep;i++){
        argx=(x-amostran[i])/h_opt;
    sump0+=erfff(argx);
}
    sump0 = sump0/nrep;
    sumpproc=1.0-sump1+sump0;

    argx=( 3.291-media_amostra)/desvio_amostra;
sump1=erfff(argx);
    argx=(-3.291-media_amostra)/desvio_amostra;
sump0=erfff(argx);
    sumpproc0s+=1.0-sump1+sump0;
    sumpproc0s2+=sqr(1.0-sump1+sump0);
    sumpproc0+=sqr(1.0-sump1+sump0- 0.001);

delta=0.01;
ise_emp=0.0;
x=-5.0;
while(x<=5.0){
    sump=0.0;
        for(i=0;i<nrep;i++){ /* empirica */
            if(x>amostran[i])sump+=1.0;
        }
        sump = sump/nrep - erfff(x);
    ise_emp+= sqr(sump);
    x+=delta;
}
ise_emp*=delta;
    printf("%g\n",ise_emp);
ise_emp_i=(int)(ise_emp*10000+0.5);
    if(ise_emp_i<1000)vetor_ise_emp[ise_emp_i]++;
    if(flagg==0)
        putp(ise_emp_i, 400 + vetor_ise_emp[ise_emp_i], 7);

```

```

gc=0.0;
for(i=0;i<=iphi2;i++){
    lamb=i*delta_lambda;
    gc+=sqr(lamb*lamb)*(vetor_phi2[i]-1.0/nrep);
}
gc*=delta_lambda/pi;
if(gc*500<1000)vetor_gc[(int)(gc*500)]++;
if(flagg)
    putp((int)(400+gc*500/3),
        (int)(270-1*(vetor_gc[(int)(gc*500)])),8);
g_opt=cte_ker_g/(gc*nrep);
g_opt=pow(g_opt,inv5);
g_opt_i=(int)(g_opt*100+0.5);
if(g_opt_i<1000)vetor_g_opt[g_opt_i]++;
printf("%g ",g_opt);
if(flagg==0)
    putp(400+g_opt_i, vetor_g_opt[g_opt_i], 9);
delta=0.01;
ise_g=0.0;
Fng=0.0;
x=-5.0; xj=0; xjmax=800;
while(x<=5.0){
    sumfg=0.0;
    for(i=0;i<nrep;i++){ /* nucleo estimador densidade em x */
        argx=(x-amostran[i])/g_opt;
        sumfg+=gauss(argx);
    }
    sumfg/=(nrep*g_opt);
    Fng+=coef_simpson(xj)*sumfg*delta/3.0;
    ise_g+= sqr(Fng-erfff(x));
    x+=delta;
    xj++;
}
ise_g*=delta;
printf("%g\n",ise_g);
ise_g_i=(int)(ise_g*10000+0.5);
if(ise_g_i<1000)vetor_ise_g[ise_g_i]++;
if(flagg==0)

```

```

        putp(400+ise_g_i, 200 + vetor_ise_g[ise_g_i], 10);

s_lambda+=(Lambda);
s2_lambda+=sqr(Lambda);
    s_hc+=hc;
    s2_hc+=sqr(hc);
    s_hopt+=h_opt;
    s2_hopt+=sqr(h_opt);
    s_ise+=ise;
    s2_ise+=sqr(ise);
    s_gc+=gc;
    s2_gc+=sqr(gc);
    s_gopt+=g_opt;
    s2_gopt+=sqr(g_opt);
    s_ise_g+=ise_g;
    s2_ise_g+=sqr(ise_g);
    s_emp+=ise_emp;
    s2_emp+=sqr(ise_emp);
    mse_h+=sqr(hc-hc0);
    mse_g+=sqr(gc-gc0);
    mse_hopt+=sqr(h_opt-h_opt0);
    mse_gopt+=sqr(g_opt-g_opt0);
s_proc+=sumpproc;
s2_proc+=sqr(sumpproc);
mse_proc+=sqr(sumpproc-proc0);

Di_emp = ise_emp - ise;
Di_g = ise_g - ise;
s_Di_emp+=Di_emp;
s2_Di_emp+=sqr(Di_emp);
s_Di_g+=Di_g;
s2_Di_g+=sqr(Di_g);
}
m_lambda = s_lambda/nexp;
d_lambda = sqrt( (s2_lambda - sqr(s_lambda) / nexp ) / (nexp-1.0) );

m_hc = s_hc/nexp;
d_hc = sqrt( (s2_hc - sqr(s_hc) / nexp ) / (nexp-1.0) );

```

```

m_hopt = s_hopt/nexp;
d_hopt = sqrt( (s2_hopt - sqr(s_hopt) / nexp ) / (nexp-1.0) );

m_ise = s_ise/nexp;
d_ise = sqrt( (s2_ise - sqr(s_ise) / nexp ) / (nexp-1.0) );

m_gc = s_gc/nexp;
d_gc = sqrt( (s2_gc - sqr(s_gc) / nexp ) / (nexp-1.0) );

m_gopt = s_gopt/nexp;
d_gopt = sqrt( (s2_gopt - sqr(s_gopt) / nexp ) / (nexp-1.0) );

m_ise_g = s_ise_g/nexp;
d_ise_g = sqrt( (s2_ise_g - sqr(s_ise_g) / nexp ) / (nexp-1.0) );

m_emp = s_emp/nexp;
d_emp = sqrt( (s2_emp - sqr(s_emp) / nexp ) / (nexp-1.0) );

m_Di_emp = s_Di_emp/nexp;
d_Di_emp = sqrt( (s2_Di_emp - sqr(s_Di_emp) / nexp ) / (nexp-1.0) );

m_Di_g = s_Di_g/nexp;
d_Di_g = sqrt( (s2_Di_g - sqr(s_Di_g) / nexp ) / (nexp-1.0) );

m_proc = s_proc/nexp;
d_proc = sqrt( (s2_proc - sqr(s_proc) / nexp ) / (nexp-1.0) );

m_proc0 = sumpproc0s/nexp;
d_proc0 = sqrt( (sumpproc0s2 - sqr(sumpproc0s) / nexp ) / (nexp-1.0) );

stud_emp = m_Di_emp * sqrt(nexp) / d_Di_emp;
stud_g = m_Di_g * sqrt(nexp) / d_Di_g;
pv_emp = 1.0 - erfff(stud_emp);
pv_g = 1.0 - erfff(stud_g);

mse_h/=nexp;
mse_g/=nexp;
mse_hopt/=nexp;
mse_gopt/=nexp;
mse_proc/=nexp;

```

```

sumpproc0/=nexp;

printf("nrep = %d   nexp = %g\n",nrep,nexp);
printf("media_lambda = %g   desv_lambda = %g\n",m_lambda,d_lambda);
printf("media_hc = %g   desv_hc = %g\n",m_hc,d_hc);
printf("media_hopt = %g   desv_hopt = %g\n",m_hopt,d_hopt);
printf("media_ise = %g   desv_ise = %g\n",m_ise,d_ise);
printf("media_gc = %g   desv_gc = %g\n",m_gc,d_gc);
printf("media_gopt = %g   desv_gopt = %g\n",m_gopt,d_gopt);
printf("media_ise_g = %g   desv_ise_g = %g\n",m_ise_g,d_ise_g);
printf("media_emp = %g   desv_emp = %g\n",m_emp,d_emp);
printf("mse_h = %g   mse_g = %g\n",mse_h,mse_g);
printf("mse_hopt = %g   mse_gopt = %g\n",mse_hopt,mse_gopt);
printf("pvalor_emp = %g   pvalor_g = %g\n",pv_emp,pv_g);
printf("stud_emp = %g   stud_g = %g\n",stud_emp,stud_g);
printf("m_Di_emp = %g   m_Di_g = %g\n",m_Di_emp,m_Di_g);
printf("d_Di_emp = %g   d_Di_g = %g\n",d_Di_emp,d_Di_g);
printf("media_proc = %g   desv_proc = %g\n",m_proc,d_proc);
printf("mse_proc = %g\n",mse_proc);
printf("sumpproc0 = %g\n",sumpproc0);
printf("media_proc0 = %g   desv_proc0 = %g\n",m_proc0,d_proc0);

fprintf(s,"nrep = %d   nexp = %g\n",nrep,nexp);
fprintf(s,"media_lambda = %g   desv_lambda = %g\n",m_lambda,d_lambda);
fprintf(s,"media_hc = %g   desv_hc = %g\n",m_hc,d_hc);
fprintf(s,"media_hopt = %g   desv_hopt = %g\n",m_hopt,d_hopt);
fprintf(s,"media_ise = %g   desv_ise = %g\n",m_ise,d_ise);
fprintf(s,"media_gc = %g   desv_gc = %g\n",m_gc,d_gc);
fprintf(s,"media_gopt = %g   desv_gopt = %g\n",m_gopt,d_gopt);
fprintf(s,"media_ise_g = %g   desv_ise_g = %g\n",m_ise_g,d_ise_g);
fprintf(s,"media_emp = %g   desv_emp = %g\n",m_emp,d_emp);
fprintf(s,"mse_h = %g   mse_g = %g\n",mse_h,mse_g);
fprintf(s,"mse_hopt = %g   mse_gopt = %g\n",mse_hopt,mse_gopt);
fprintf(s,"pvalor_emp = %g   pvalor_g = %g\n",pv_emp,pv_g);
fprintf(s,"stud_emp = %g   stud_g = %g\n",stud_emp,stud_g);
fprintf(s,"m_Di_emp = %g   m_Di_g = %g\n",m_Di_emp,m_Di_g);
fprintf(s,"d_Di_emp = %g   d_Di_g = %g\n",d_Di_emp,d_Di_g);
fprintf(s,"media_proc = %g   desv_proc = %g\n",m_proc,d_proc);
fprintf(s,"mse_proc = %g\n",mse_proc);
fprintf(s,"sumpproc0 = %g\n",sumpproc0);

```

```

fprintf(s,"media_proc0 = %g    desv_proc0 = %g\n",m_proc0,d_proc0);

fim;;
fclose(anormal);
fclose(s);
fclose(vetor);
printf("fim do programa\n");
}

float alea(long idum0){
    long k;
    float ans;
    idum0 ^= MASK;
    k=(idum0)/IQ;
    idum0=IA*(idum0-k*IQ)-IR*k;
    if(idum0<0) idum0+=IM;
    ans=AM*(idum0);
    idum0^=MASK;
    idum=idum0;
    return(ans);
}

double sqr(double x){
    return(x*x);
}

double pow(double x, double y){ /* x**y */
    return(exp(y*log(x)));
}

double coef_simpson(int i){
    if((i==0)|| (i==xjmax))return(1.0);
    else if(i%2==0)return(2.0);
    else return(4.0);
}

double normal2(double media, double desv){
    double u1,u2,v1,v2,s,rr,x;
    if(normal_impar==0){
        do{

```

```

        u1=alea(idum);
        u2=alea(idum);
        v1=2*u1-1;
        v2=2*u2-1;
        s=v1*v1+v2*v2;
    }while(s>1);
    rr=sqrt((-2*log(s))/s);
    x=rr*v1;
    normal_y=rr*v2;
    normal_impar=1;
    return(x*desv+media);
}
else{
    normal_impar=0;
    return(normal_y*desv+media);
}
}

double teste_ad(){ /* Teste de Normalidade de Anderson-Darling */
    double termo,soma,media,dp,aux1,aux2,pp[1000];
    int i,j;
    media=aux2=0.0;
    for(i=0;i<nrep;i++){
        media+=amostran[i];
        aux2+=sqr(amostran[i]);
    }
    media/=nrep;
    dp=sqrt((aux2-nrep*sqr(media))/(nrep-1.0));
    for(i=0;i<nrep;i++){
        pp[i]=erfff((amostran[i]-media)/dp);
    }
    termo=0.0;
    for(i=1;i<=nrep;i++){
        termo+=(2.0*i-1.0)*(log(pp[i-1])+log(1.0-pp[nrep+1-i-1]));
    }
    termo/=nrep;
    termo=-nrep-termo;
    return(termo);
}

```

```

double erfff(float x){/* integral da funcao de Gauss de -inf. ate' x */
    float t,z,ans;
    z=fabs(x);
    z/=sqrt(2.0);
    t=1.0/(1.0+0.5*z);
    ans=t*exp(-z*z
-1.26551223
+t*(1.00002368
+t*(.37409196
+t*(.09678418
+t*(-.18628806
+t*(.27886807
+t*(-1.13520398
+t*(1.48851587
+t*(-.82215223
+t*.17087277
))))))));
    return x>=0.0? 1.0-ans/2.0 : ans/2.0;
}
double gauss(double x){
    return(exp(-x*x/2.0)/cgauss);
}

```


Rotina para Geração de Números Aleatórios - Distribuição Gama:

```
/* Algoritmo Geracao Numeros Aleatorios - Gamma */
/*****/

float gamdev(int ia){
    int j;
    float am,e,s,v1,v2,x,y;

    if(ia<1) printf("Error in routine gamdev\n");
    if(ia<6){
        x=1.0;
        for(j=1;j<=ia;j++) x*=alea(idum);
        x=-log(x);
    }
    else{
        do{
            do{
                do{
                    v1=alea(idum);
                    v2=2.0*alea(idum)-1.0;
                }while(v1*v1+v2*v2>1.0);
                y=v2/v1;
                am=ia-1;
                s=sqrt(2.0*am+1.0);
                x=s*y+am;
            }while(x<=0);
            e=(1.0+y*y)*exp(am*log(x/am)-s*y);
        }while (alea(idum)>e);
    }
    return x;
}

double integral_gamma2(double x){
    return( 1.0 - (1.0+x)*exp(-x) );
}

double densidade_gamma2(double x){
    return( x*exp(-x) );
}
```

Rotina para Geração de Números Aleatórios - Distribuição Bimodal:

```
/* Algoritmo Geracao Numeros Aleatorios - Bimodal */
/*****/

for(jj=0;jj<nrep;jj++){/*geracao de uma amostra de nrep valores normais*/
    if(alea(idum)>0.5)x=normal2(-1.5,1);
    else x=normal2(1.5,1);
    amostran[jj]=x;
    media_amostra+=x;
    media_amostra2+=sqr(x);
    fprintf(anormal,"%g\n",x);
    quiq=(x+5)/cgraf;
    xi=(int)(tamv*quiq+0.5); if((xi<1000)&&(xi>=0))fdp[xi]++;
    if(flagg)putp(xi,(int)((470-(fdp[xi])/1.0)),2);
}
```