UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CLÁUDIA THEIS DA SILVEIRA

IMPLEMENTAÇÃO E COMPARAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE PARÂMETROS DE RTN

Porto Alegre
2021

CLÁUDIA THEIS DA SILVEIRA

IMPLEMENTAÇÃO E COMPARAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE PARÂMETROS DE RTN

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Engenharia da Computação.

ORIENTADOR: Prof. Dr. Gilson Inácio Wirth

Porto Alegre

2021

CLÁUDIA THEIS DA SILVEIRA

IMPLEMENTAÇÃO E COMPARAÇÃO DE ALGORITMOS PARA EXTRAÇÃO DE PARÂMETROS RTN

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Gilson Inácio Wirth, UFRGS

Doutor pela Universitaet Dortmund – Dortmund, Alemanha

Banca Examinadora:

Prof(a). Dr(a). Fernanda Lima Kastensmidt, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Altamiro Amadeu Susin, UFRGS

Doutor pela Institut National Polytechnique de Grenoble – Grenoble, França

Prof. Dr. Ivan Muller, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Coordenador do PPGEE: _____

Prof. Dr. Sérgio Luís Haffner

Porto Alegre, Abril de 2021.

DEDICATÓRIA

Dedico este trabalho a Deus, por ser essencial em minha vida, por sempre acalmar minhas angústias e minhas preocupações, e por me dar forças pra chegar até aqui. Aos meus pais, Gilberto Rodrigues da Silveira e Iolanda Theis da Silveira, por toda dedicação e esforços empregados na minha educação, e por me ajudarem a alcançar meus objetivos. A minha irmã, Márcia da Silveira Blanco, por todo apoio e carinho demonstrado durante esta caminhada. Ao meu irmão, André Theis da Silveira, a minha família e aos amigos, que de uma forma ou outra, sempre me ajudaram e me motivaram a seguir em frente.

AGRADECIMENTOS

Agradeço, primeiramente, ao Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pela oportunidade de realização de trabalhos em minha área de pesquisa.

Agradeço, ao meu orientador, Prof. Dr. Gilson Inácio Wirth, pelos conhecimentos transmitidos durante o mestrado, por toda dedicação e auxílio na confecção desta dissertação.

Agradeço, aos colegas do PPGEE pelo seu auxílio nas tarefas desenvolvidas durante o curso e apoio na revisão deste trabalho.

Agradeço, ao CNPQ pelo suporte financeiro concedido durante a realização do mestrado.

Agradeço, aos colegas do laboratório LAPROT, por todo apoio e incentivo durante a realização deste trabalho.

Agradeço, a todos os professores que contribuíram para minha formação e a Universidade Federal do Rio Grande do Sul, UFRGS, pelo ensino de qualidade.

Agradeço, aos meus amigos, por toda a cumplicidade e apoio durante a elaboração deste trabalho.

Por fim, um agradecimento especial aos meus pais, Gilberto e Iolanda; a minha irmã, Márcia; ao meu irmão, André; e ao meu namorado, Guilherme Acosta, pelo apoio e paciência durante a realização deste trabalho.

RESUMO

Atualmente o estudo de ruído gerado internamente pelos dispositivos, como o *Random Telegraph Noise* (RTN), é de grande relevância, visto que este estudo pode fornecer informações importantes sobre as propriedades físicas e atomísticas dos dispositivos micro e nano eletrônicos, dentre os quais como *Resistive Random Access Memory* (ReRAM) e MOSFET. Neste trabalho, será desenvolvido um algoritmo baseado em uma ferramenta conhecida como *Hidden Markov Model* (HMM), que é uma técnica muito utilizada na análise de sinais estocásticos. Com o desenvolvimento deste algoritmo pretende-se realizar a extração de parâmetros de sinais RTN a partir de valores sintéticos e dados experimentais medidos em dispositivos eletrônicos, tais como a ReRAM e o MOSFET. Além disso, será realizada uma comparação dos resultados extraídos pelo método desenvolvido com os resultados obtidos através de um segundo método de extração de parâmetros de sinais RTN, o qual foi implementado pelo aluno Pedro Augusto Böckmann Alves, e se baseia na discretização de medidas. Por fim, através da comparação dos resultados extraídos por cada um dos métodos, será feita uma análise de desempenho de ambos os algoritmos implementados, na presença de ruído Gaussiano (branco). Posteriormente, os algoritmos são aplicados para a extração de parâmetros de RTN a partir de dados experimentais de RTN medidos em dispositivos MOSFETs e ReRAMs. O parâmetro medido neste trabalho foi a corrente elétrica ao longo do tempo. Os resultados deste trabalho mostraram que ambos os métodos são capazes de realizar a extração dos parâmetros de sinais RTN, e também pode-se observar que o segundo método é menos preciso quando comparado ao método baseado na ferramenta HMM.

Palavras-chave: Algoritmo. *Random Telegraph Noise. Hidden Markov Model*. Extração de Parâmetros.

ABSTRACT

Nowadays the study of noise generated internally by devices, such as the Random Telegraph Noise (RTN), is of great relevance, since this study can provide important information about the physical and atomistic properties of micro and nano electronic devices, among which as Resistive Random Access Memory (ReRAM) and MOSFET. In this work, an algorithm based on a tool known as the Hidden Markov Model (HMM) will be developed, which is a technique widely used in the analysis of stochastic signals. From the development of this algorithm it is intended to perform the extraction of RTN signal parameters from synthetic values and experimental data measured in electronic devices, such as ReRAM and MOSFET. In addition, a comparison of the results extracted by the developed method with the results obtained through a second method of extracting parameters from RTN signals will be carried out, which was implemented by the student Pedro Augusto Böckmann Alves, and is based on the discretization of measures. Finally, by comparing the results extracted by each method, a performance analysis of both implemented algorithms will be made, in the presence of Gaussian (white) noise. Subsequently, the algorithms are applied to extract parameters from experimental RTN data measured in MOSFETs and ReRAMs. The parameter measured in this work was the electric current over time. The results of this work showed that both methods are capable of extracting the parameters of RTN signals, and it can also be observed that the second method is less accurate when compared to the method based on the HMM tool.

Keywords: Algorithm. Random Telegraph Noise. Hidden Markov Model. Parameter extraction.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

aRTN *Anomalous Random Telegraph Noise*

EM *Expectation-Maximization*

FHMM *Factorial Hidden Markov Model*

HMM *Hidden Markov Model*

HRS *High Resistance State*

LRS *Low Resistance State*

MOSFET *Metal Oxide Field Effect Transistor*

mRTN *Mutant Random Telegraph Noise*

PSD *Power Spectrum Density*

RERAM *Resistive Random Access Memory*

RTN *Random Telegraph Noise*

RTS *Random Telegraph Signal*

TLP *Time-Lag Plot*

tRTN *Temporary Random Telegraph Noise*

SUMÁRIO

1 INTRODUÇÃO

O ruído sempre foi uma característica notavelmente importante dos dispositivos eletrônicos e, por várias décadas, foi olhado de muitas perspectivas diferentes. O próprio ruído, nas suas várias formas, sempre foi simultaneamente um fenômeno fundamental fascinante, uma ferramenta versátil para avaliar a imperfeição dos materiais e a qualidade dos processos de produção dos dispositivos e uma séria preocupação do ponto de vista da confiabilidade do circuito (PUGLISI, 2014).

O ruído de baixa frequência é um limitador de desempenho em circuitos analógicos, digitais e de radiofrequência, introduzindo ruído de fase em osciladores e reduzindo a estabilidade de células SRAM, por exemplo (BOTH, 2017).

Com o avanço da tecnologia e o constante processo de miniaturização dos dispositivos eletrônicos, o principal tipo de ruído de interesse em tais dispositivos é o ruído de baixa frequência, principalmente na forma de *"Flicker"* (1/f) e *Random Telegraph Noise* (RTN), visto como um fenômeno prejudicial que limita a confiabilidade. O estudo do ruído RTN é de grande importância, pois este ruído é frequentemente observado em uma grande variedade de dispositivos, como por exemplo, a *Resistive Random Access Memory (ReRAM)* e o MOSFET (PUGLISI, 2014; SEO, 2017).

A *ReRAM* é um dispositivo de memória, o qual sua resistência varia dependendo das condições de condução e do seu histórico de operação. Tipicamente, a *ReRAM* possui dois estados estáveis e dois eventos de transição. Os estados estáveis são definidos como o estado de baixa resistência (LRS) e o estado de alta resistência (HRS). E os eventos de transição são definidos como o estado de SET e o estado de RESET (PRAKASH, 2016, PUGLISI, 2018).

O RTN é o tipo de ruído mais importante em dispositivos *ReRAM*. A análise deste ruído pode contribuir com informações essenciais sobre a física destes dispositivos e explicar

o papel e as propriedades atomísticas dos defeitos envolvidos na geração do RTN. Sua origem é geralmente atribuída à captura e emissão de portadores de carga pelos defeitos no dispositivo. Estes defeitos são denominados armadilhas ou *traps*. A análise estatística destes fenômenos pode ser a ferramenta ideal para compreender, na prática, os efeitos destes eventos durante o mecanismo de comutação resistiva (PUGLISI, 2013a).

Existem algumas técnicas de análise que possibilitam o estudo dos sinais RTN e a recuperação de suas propriedades estatísticas, como por exemplo, algoritmos que permitem a extração dos parâmetros do sinal RTN a partir dos dados medidos (PUGLISI, 2014).

Uma das técnicas utilizadas para a avaliação estatística de um sinal RTN é uma abordagem muito popular e versátil, conhecida como *Hidden Markov Model* (HMM), a qual é uma ferramenta poderosa comumente utilizada em reconhecimento de padrões e análise de sinais estocásticos (PUGLISI, 2014).

Neste sentido, o presente trabalho propõe o desenvolvimento de um algoritmo, baseado na técnica de HMM, que seja capaz de extrair os parâmetros de sinais RTN, mesmo na presença de outras perturbações, como o ruído Gaussiano. Dessa forma, espera-se disponibilizar as informações necessárias para a caracterização deste ruído. Além disso, espera-se realizar uma comparação com o método desenvolvido e um segundo método, o qual foi implementado pelo aluno Pedro Augusto Böckmann Alves. O segundo método se baseia na discretização de medidas, e tem com objetivo isolar o RTN do seu sinal de origem, e também caracteriza-lo para interpretações e análises futuras. Neste trabalho, o primeiro método será denominado método A, e o segundo método será chamado de método B.

A organização desta dissertação está disposta como segue:

O capítulo 2 apresenta uma revisão sobre o ruído RTN, subdividindo-o na apresentação de suas características estatísticas, e a classificação dos sinais RTN. Além disso, o capítulo também discorre sobre as principais ferramentas de análise destes sinais, como o

histograma e *Time-Lag Plot* (TLP), o HMM e o FHMM (*Factorial Hidden Markov Model*). Por fim, são apresentados os problemas básicos de um sistema HMM e suas soluções.

O capítulo 3 descreve os métodos utilizados na extração dos parâmetros de um RTN. Em um primeiro momento é explicado o funcionamento do algoritmo que gera a série temporal de um RTN sintético. Em seguida, é apresentado o método de extração de parâmetros desenvolvido neste trabalho (método A), o qual se baseia em um sistema HMM. Finalmente, é descrito o segundo método de extração implementado com base na discretização de medidas (método B).

O capítulo 4 apresenta os resultados deste trabalho. Em um primeiro momento mostram-se os resultados referentes à geração de um RTN sintético, e em seguida é realizada a extração dos parâmetros deste RTN utilizando os dois métodos apresentados no capítulo anterior, com o objetivo final de avaliar o funcionamento de ambos para o que foi proposto.

No capítulo 5, são apresentadas as conclusões gerais a respeito do trabalho proposto, como também são dadas algumas sugestões para continuidade deste trabalho e para trabalhos futuros a serem realizados nesta área de pesquisa.

2 REVISÃO DA LITERATURA

Neste capítulo, são abordados e discutidos conceitos relevantes, com o objetivo de situar o trabalho dentro de sua área de pesquisa e aprofundar o conhecimento sobre o tema.

2.1 Random Telegraph Noise (RTN)

O RTN é um tipo de ruído que é frequentemente observado em uma grande variedade de dispositivos semicondutores. Sua origem é geralmente atribuída à captura e emissão de carga pelos defeitos no dispositivo (PUGLISI, 2016). Estes defeitos são constantemente chamados de armadilhas (do inglês, *traps*).

Segundo (PUGLISI, 2020), a análise das características do RTN é uma ferramenta fundamental para obter informações importantes sobre as propriedades das armadilhas e do próprio dispositivo.

O estudo dos sinais RTN apresentado nesta seção inicia com a análise de suas características estatísticas, e em seguida, são apresentados os tipos de sinais RTN encontrados na literatura.

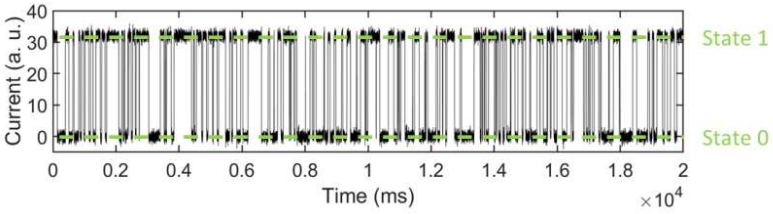
2.1.1 Características Estatísticas dos Sinais RTN

Em sua forma mais simples, um sinal RTN é associado com a comutação aleatória de uma quantidade observável entre dois estados discretos, como por exemplo, a corrente. Estes estados podem ser denominados como estado 1 e estado 0, conforme mostrado na Figura 1 (PUGLISI, 2020).

A probabilidade por unidade de tempo para uma transição ocorrer do estado 0 para o estado 1 é dada por $1/\tau_0$. E a probabilidade da transição do estado 1 para o estado 0 ocorrer é dada por $1/\tau_1$. Isso implica que o tempo gasto em qualquer um dos estados segue uma distribuição exponencial. De fato, a probabilidade para o sistema, inicialmente assumido em um estado bem definido (0 ou 1) no tempo t=0, permanecer neste mesmo estado (enquanto

nenhuma transição ocorrer) no tempo t pode ser expressa como nas Equações 1 e 2 (PUGLISI, 2020).

Figura 1 - Exemplo de um sinal RTN simples de dois níveis.



Fonte: (PUGLISI, 2020).

$$p_0(t) = \frac{1}{\tau_0} e^{(\frac{-t}{\tau_0})} \quad (1)$$

$$p_1(t) = \frac{1}{\tau_1} e^{(\frac{-t}{\tau_1})} \quad (2)$$

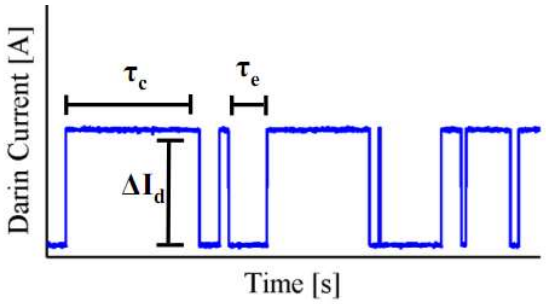
O sinal RTN de dois níveis é estatisticamente caracterizado por três parâmetros: o tempo médio gasto em cada um dos dois estados ($\tau_0 \; e \; \tau_1$), e a amplitude da flutuação do RTN (ΔI, como por exemplo, a diferença entre os níveis discretos de corrente). Estes parâmetros podem ser estimados a partir de dados experimentais (PUGLISI, 2020; SILVA, 2016).

Os parâmetros do RTN geralmente dependem das condições de operação e da física dos dispositivos. Assim, é possível estabelecer uma ligação entre as propriedades do sinal e as propriedades físicas dos defeitos associados com o RTN. Portanto, já que as transições entre os níveis discretos são associados com a captura de carga a partir de uma armadilha e a emissão de carga por uma armadilha, τ_1 e τ_0 são usualmente renomeados como tempo de captura (τ_c) e tempo de emissão (τ_e) (PUGLISI, 2020).

Logo, os dois níveis discretos, entre os quais ocorre a flutuação do sinal, representam o estado da armadilha (ocupada ou vazia). A Figura 2 ilustra um RTN no domínio do tempo, onde uma corrente de dreno flutua entre dois níveis fixos com tempos estocásticos de baixo e

alto nível, assemelhando-se a um *Random Telegraph Signal* (RTS) (PUGLISI, 2020; SILVA, 2016).

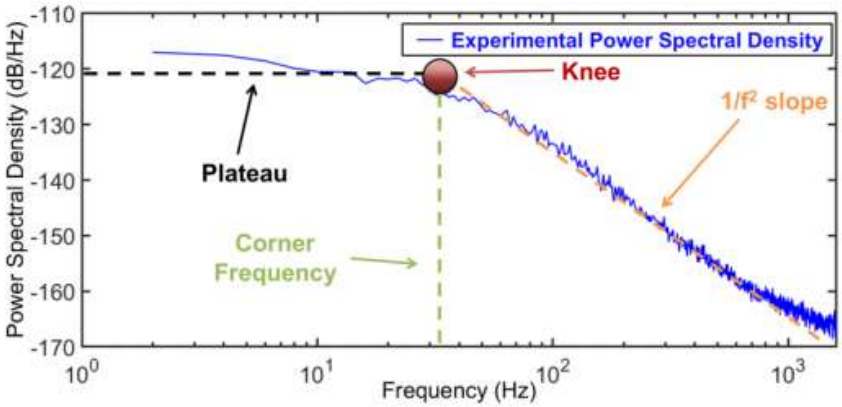
Figura 2 - RTS causado por uma única armadilha.



Fonte: (SILVA, 2016).

No domínio da frequência, o RTN associado com uma única armadilha é representado por uma função Lorentziana, a qual na escala log-log é caracterizada por uma região de platô em frequências relativamente baixas ($2\pi f \ll 1/\tau_e + 1/\tau_c$), um joelho definido por uma frequência de corte ($2\pi f_c = 1/\tau_e + 1/\tau_c$) e uma região de inclinação negativa dada por $1/f^2$. A Figura 3 ilustra o RTN no domínio da frequência (PUGLISI, 2020; SILVA, 2016).

Figura 3 - Espectro RTN no domínio frequência, em escala log-log.



Fonte: (PUGLISI, 2020).

A densidade do espectro de potência (PSD) do RTN, dado por uma única armadilha, é calculado como mostra a Equação 3, onde τ_e e τ_c são a média dos tempos de emissão e captura, respectivamente, e ΔI é a amplitude da flutuação de corrente (SILVA, 2016).

$$S(f) = \frac{4(\Delta I)^2}{(\tau_e + \tau_c) * [\left(\frac{1}{\tau_e} + \frac{1}{\tau_c}\right)^2 + (2\pi f)^2]} \quad (3)$$

Tipicamente, um sinal RTN de dois níveis é associado com a captura de carga por uma única armadilha e a emissão de carga a partir desta armadilha. Os sinais RTN multi-níveis, em vez disso, são associados com a captura e emissão de carga ocorrendo em mais de uma armadilha. No domínio da frequência, a soma de processos RTN independentes é dada pela superposição dos espectros Lorentzianos correspondentes. Foi demonstrado que em condições moderadas sobre o número de espectros Lorentzianos sobrepostos, o espectro geral tende para a forma da Equação 4. Isto é, se o número de espectros Lorentzianos não for muito baixo e se seus valores de frequência de corte não forem muito próximos uns dos outros (PUGLISI, 2020; SILVA, 2016).

$$S(f) = \frac{A}{f} \quad (4)$$

Onde A é um parâmetro que representa a densidade espectral de potência na frequência unitária (PUGLISI, 2020).

2.1.2 Classificação dos Sinais RTN

Os sinais RTN podem ser catalogados de acordo com várias classificações. Um dos critérios poderia ser simplesmente o número de níveis discretos que o sinal exibe. Em sua forma mais simples, um sinal RTN aparece como a comutação aleatória de uma quantidade

observável (tensão, corrente, impedância) entre dois estados discretos. Já os sinais RTN multi-níveis são caracterizados por mais de dois níveis discretos (PUGLISI, 2020).

O sinal RTN multi-nível pode ser visto como a superposição de um número de sinais RTN de dois níveis, com cada um deles sendo associado com a captura e emissão de carga por uma armadilha individual, desde que os processos de captura e emissão em tais armadilhas possam ser considerados mutuamente independentes. Neste caso, o sinal RTN multi-nível observado resulta da superposição de vários sinais RTN de dois níveis (componentes), cada um deles relacionado a uma armadilha individual (PUGLISI, 2020).

Existem ainda outras classificações para o sinal RTN, como o RTN anômalo (aRTN) (PUGLISI, 2015). Assim como o RTN multi-nível, o aRTN também é caracterizado por mais de dois níveis discretos, porém, o aRTN não pode ser modelado como uma superposição de flutuações de RTN independentes de dois níveis. Os componentes RTN de dois níveis que somam o sinal aRTN observado possuem um grau de correlação, ou seja, as características de um componente em um determinado instante de tempo são determinadas também pelo estado em que os outros componentes são encontrados (PUGLISI, 2020).

Além das classificações já citadas, existem alguns fenômenos temporários, os quais são chamados de RTN temporário (tRTN), e RTN mutante (mRTN). O sinal tRTN é uma flutuação RTN de dois níveis que aparece e desaparece aleatoriamente durante as medições do RTN. E o mRTN é uma flutuação RTN de dois níveis que muda suas características estatísticas de forma aleatória e temporária (PUGLISI, 2020).

2.2 Ferramentas de Análise dos Sinais RTN

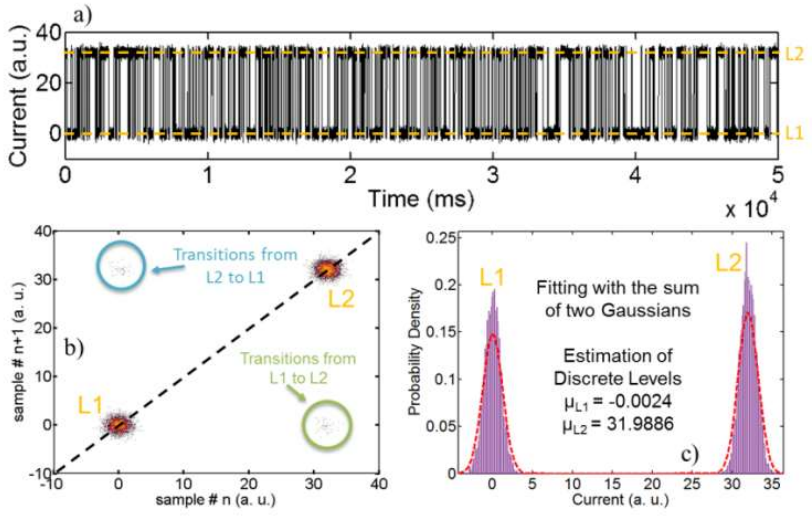
Existem técnicas de análise que possibilitam o estudo dos sinais RTN e a recuperação de suas propriedades estatísticas, como por exemplo, algoritmos que permitem a extração dos parâmetros do sinal RTN a partir dos dados medidos. Nesta seção, será apresentada uma

revisão dos métodos mais comuns utilizados, citando as vantagens e desvantagens de cada metodologia.

2.2.1 Histograma e Time-Lag Plots (TLP)

A caracterização dos dados RTN visa extrair os parâmetros do sinal a partir de dados experimentais não processados. No caso simples de um RTN de dois níveis, como mostrado na Figura 4(a), uma abordagem representativa frequentemente usada na literatura se baseia no ajuste do histograma do sinal RTN medido, como mostrado na Figura 4(c). Nesta abordagem, é realizado o ajuste do histograma resultante com um modelo de distribuição bi-Gaussiana, assim é possível estimar os dois níveis discretos (ou seja, os valores médios das duas distribuições Gaussianas). A principal suposição deste procedimento é que o ruído de fundo devido a fontes de ruído adicionais e o ruído introduzido pela instrumentação de medição podem ser assumidos como Gaussianos. Outra ferramenta que pode ser utilizada na análise do sinal RTN (juntamente ou no lugar do histograma) é o *Time-Lag Plot* (TLP), mostrado na Figura 4(b) (PUGLISI, 2020).

Figura 4 - (a) RTN de dois níveis. (b) Representação em Time-Lag Plot. (c) Representação por Histograma.



Fonte: (PUGLISI, 2020).

O método TLP é frequentemente utilizado com o objetivo de avaliar a aleatoriedade de uma série temporal. TLPs podem ser usados para estimar o número de níveis discretos (e seus valores) contando o número de pontos (e suas posições) na diagonal principal do gráfico (as transições são representadas por nuvens fora da diagonal). Então, cada ponto do sinal pode ser associado a um dos dois níveis discretos e as diferentes realizações dos tempos de captura e emissão podem ser estimadas. Seus valores médios são facilmente extraídos como a média das distribuições exponenciais correspondentes (PUGLISI, 2017).

TLPs e histogramas também podem ser utilizados na extração das características de sinais RTN multi-níveis. No entanto, neste caso, é um desafio recuperar as propriedades estatísticas (ΔI, τc e τe) de cada componente individual do RTN multi-nível medido. Por exemplo, em um sinal RTN de quatro níveis, estes métodos conseguem identificar quantos níveis distintos compõem o sinal. Porém, é complicado (embora tecnicamente possível) traduzir os quatro níveis discretos nas duas amplitudes (ΔI) dos componentes RTN sobrepostos. Da mesma forma, embora seja fácil determinar o tempo médio gasto em cada estado, é difícil converter essas informações nos tempos de captura e emissão dos dois componentes RTN (PUGLISI, 2020).

2.2.2 Hidden Markov Model (HMM)

Existem algumas metodologias mais avançadas, as quais exploram a natureza Markoviana dos sinais RTN com o objetivo de avaliar seus parâmetros estatísticos. Uma das técnicas utilizadas para esta avaliação é uma abordagem muito popular e versátil, conhecida como *Hidden Markov Model (HMM)*, a qual é uma ferramenta poderosa comumente utilizada em reconhecimento de padrões e análise de sinais estocásticos (PUGLISI, 2020).

Em HMM, o sistema é assumido como um processo de Markov (isto é, sem memória) com estados escondidos, tais como, os níveis discretos do RTN. Cada estado é associado com um conjunto de probabilidades de transição definindo o quão provável é para o sistema, estando em um dado estado em um dado instante de tempo, comutar para outro estado possível (incluindo ele mesmo) no instante de tempo seguinte (PUGLISI, 2020).

Em um HMM, uma sequência de observações, $\{Y_t\}$ com t = 1,...,T, é modelada especificando uma relação probabilística entre as observações e o conjunto de estados escondidos (desconhecidos) $\{S_t\}$ através de uma estrutura de transição de Markov ligando os estados. Nesta situação, o estado é representado por uma variável aleatória assumindo um em cada N valores em cada instante de tempo. A aproximação HMM depende de duas suposições de independência condicional (PUGLISI, 2014; PUGLISI, 2020):

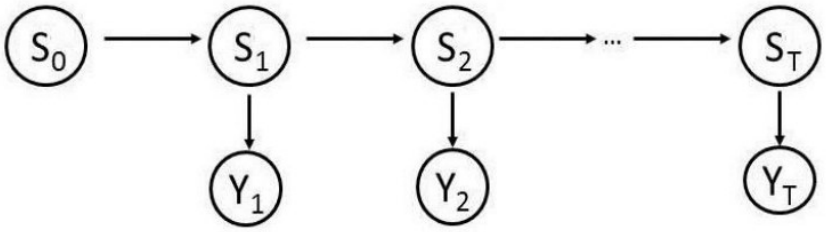
- S_t depende somente de S_{t-1} (conhecido como propriedade de primeira ordem de Markov ou propriedade "sem memória");
- Y_t é independente de todas as outras observações, dado o estado S_t.

A probabilidade conjunta para a sequência de estados e observações pode ser formalizada como na Equação 5.

$$P(S_t|Y_t) = P(S_1) \,.P(S_1|Y_1).\prod_{t=2}^{T} P(S_t|S_{t-1}).P(Y_t|S_t) \tag{5}$$

Uma representação esquemática do HMM pode ser visualizada na Figura 5, onde a propriedade de Markov está evidenciada. Em cada instante de tempo *t*, cada saída Y_t está relacionada apenas com o estado S_t atual da cadeia de Markov que define o modelo (PUGLISI, 2014).

Figura 5 - Representação Gráfica de um HMM.



Fonte: (PUGLISI, 2020).

Segundo (RABINER, 1989), um HMM pode ser completamente definido através de cinco variáveis, sendo elas: *N*, *M*, *A*, *B* e π. A variável *N* é o número de estados desconhecidos, *S*, no modelo (por exemplo: o número de níveis discretos de corrente a serem encontrados no RTN); *M* é definido como o número de símbolos observáveis diferentes (por exemplo: os possíveis valores de corrente assumidos pelo RTN). *A* é uma matrix *N*-por-*N* definindo as probabilidades de transição entre os estados e *B* é uma matrix *N*-por-*M* definindo a probabilidade de observação de cada símbolo observável em cada estado escondido; e π é um vetor definindo a distribuição de probabilidade do estado inicial.

Logo, é possível relacionar o trabalho a ser desenvolvido com as variáveis descritas. Considerando a existência de apenas uma armadilha no material, a variável *N* será 2, ou seja, existem dois estados escondidos no modelo. Estes estados assumem os valores de 1 (para armadilha ocupada) e 0 (para armadilha desocupada). Quando o material apresenta mais de uma armadilha, existirão sempre dois estados escondidos para cada uma destas armadilhas, ou seja, para duas armadilhas tem-se *N*=4, para três armadilhas tem-se *N*=6, e assim sucessivamente. Consequentemente, pode-se generalizar *N* através da Equação 6, onde N_t é o número de armadilhas existentes no sistema.

$$N = 2 * N_t \tag{6}$$

A variável *M* é definida como sendo o número de valores observáveis distintos do modelo, e possui uma relação com o número de armadilhas dada pela Equação 7. Então, para uma armadilha tem-se dois valores observáveis, para duas armadilhas obtém-se 4 valores observáveis, para três armadilhas serão 8 valores observáveis, e assim sucessivamente.

$$M = 2^{N_t} \tag{7}$$

A matriz *A* para uma armadilha é dada pela Equação 8. A probabilidade de transição do estado 1 para o estado 0 é dada por $1/\tau_c$, e a probabilidade de transição do estado 0 para o estado 1 é dada por $1/\tau_e$. No entanto, é possível que o sistema comute para o mesmo estado em que já se encontra, ou seja, a probabilidade do sistema transicionar do estado 1 para o próprio estado 1 em um instante de tempo seguinte é dado por 1-$1/\tau_c$. O mesmo acontece com o estado 0, onde a probabilidade de transição é dada por 1-$1/\tau_e$.

$$A = \begin{bmatrix} 1 - 1/\tau_c & 1/\tau_c \\ 1/\tau_e & 1 - 1/\tau_e \end{bmatrix} \tag{8}$$

A matriz *B* define a probabilidade de observação de cada símbolo observável, em cada um dos estados escondidos do modelo estudado. Para o caso de uma armadilha, onde se trabalha com dois estados (1 e 0), haverá dois símbolos observáveis distintos no sinal de saída, os quais são gerados por esta armadilha. Estes valores podem ser definidos como v_1 e v_2, e para o caso em questão, pode-se afirmar que um dos valores (por exemplo, v_1) é gerado quando o sistema encontra-se no estado 1 (armadilha ocupada), e o outro valor (v_2) é gerado quando o sistema encontra-se no estado 0 (armadilha desocupada).

Assim, a probabilidade do símbolo observável v_1 ser gerado pelo estado 1 é 1, e a probabilidade do símbolo v_2 ser gerado pelo mesmo estado 1 é 0. Logo, a probabilidade de v_2

ser gerado pelo estado 0 é 1 e a probabilidade de v_1 ser gerado pelo estado 0 é 0. Assim, a matriz B para uma armadilha é dada pela Equação 9, onde a primeira linha da matriz representa o estado 1 e a segunda linha da matriz representa o estado 0. A primeira coluna desta matriz mostra os valores de probabilidade para o símbolo v_1, e na segunda coluna pode-se visualizar os valores de probabilidade para o símbolo v_2, em cada um dos estados.

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

O vetor π fornece a distribuição da probabilidade inicial, ou seja, a probabilidade de iniciar em cada um dos estados escondidos. No caso de uma armadilha, pode-se assumir 50% de chance do modelo iniciar em qualquer um dos estados. Neste caso, π = [0.5,0.5].

O problema de inferência deste modelo consiste em encontrar o conjunto mais provável de probabilidade dos estados escondidos, dado as observações. Isto é alcançado através de uma estimativa de probabilidade máxima dos parâmetros do HMM, considerando as observações usando o algoritmo *Forward-Backward*, que também é conhecido como algoritmo *Baum-Welch*. Então, a sequência mais provável de estados escondidos representando a dinâmica das observações pode ser atingida através do algoritmo *Viterbi*, que é um paradigma de programação dinâmico. Como resultado, a análise HMM pode eficientemente estimar os níveis de corrente discretos e a melhor sequência de estados representando os dados do RTN (PUGLISI, 2020; ESPINDOLA, 2009).

Contudo, a abordagem HMM possui algumas limitações na caracterização de sinais RTN multi-níveis. Neste caso, a rotina HMM identifica corretamente os níveis escondidos do sinal RTN gerado, e também sua sequência. É possível, ainda, estimar a duração dos intervalos de tempo que o sinal gasta em cada um dos estados, assim como seus valores médios. Porém, ao contrário de um sinal RTN de dois níveis, é impossível resolver o tempo

médio de permanência e a amplitude de flutuação das cadeias independentes de Markov que compõem o sinal (PUGLISI, 2020).

Portanto, mesmo sendo eficaz na captura da dinâmica de Markov do RTN, o HMM não é o mais adequado para caracterizar o RTN de vários níveis. Embora a análise HMM identifique corretamente os estados ocultos do RTN multi-nível e sua sequência mais provável, geralmente é impossível definir separadamente as amplitudes das flutuações e os tempos de permanência para cada armadilha única que contribui para o RTN. Isto implica que, embora a caracterização do sinal RTN seja alcançada, é impossível recuperar as características distintivas de cada armadilha que contribui para o sinal observado. Essa limitação pode ser superada usando um conceito baseado no HMM mais elaborado, conhecido como *Factorial Hidden Markov Model (FHMM)* (PUGLISI, 2020).

2.2.3 Factorial Hidden Markov Model (FHMM)

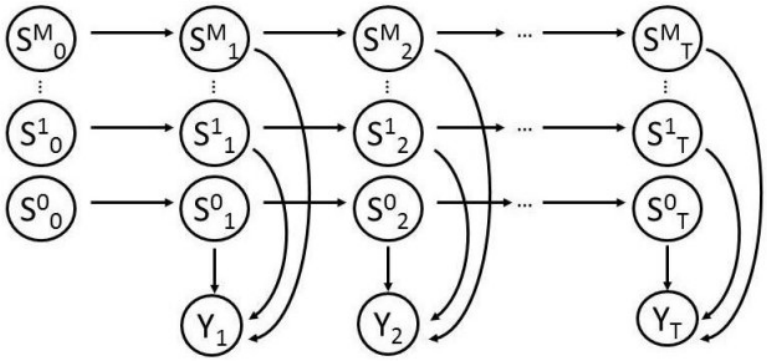
O método conhecido como FHMM é uma extensão do HMM. Este método considera o estado oculto como uma coleção de variáveis de estado *K*, ao invés de uma única variável aleatória, cada um assumindo potencialmente um dos *N* valores a cada instante (isto é: *K* cadeias de Markov diferentes e paralelas). Isso resulta em um estado espacial com uma dimensão de N^K. Se nenhuma restrição for aplicada ao modelo, ele poderá levar em consideração todas as interdependências possíveis entre as *K* cadeias de Markov, resultando em uma alta carga computacional. No entanto, uma abordagem natural consiste em assumir que cada uma das *K* cadeias de Markov evolui independentemente das outras cadeias, resultando em uma redução significativa da complexidade do problema (GHAHRAMANI, 1997; PUGLISI, 2020). Isto pode ser formalizado através da Equação 10.

$$P(S_t|S_{t-1}) = \prod_{k=1}^{K} P\left(S_t{}^k|S_{t-1}{}^k\right) \quad (10)$$

Essa também é a representação mais adequada de um RTN multi-nível, visto como uma superposição de vários RTNs de dois níveis. Essa suposição restringe cada estado da cadeia de Markov a assumir apenas um dos dois valores em cada instante de tempo (que é *N*=2). Uma representação do conceito FHMM é dada na Figura 6, onde S_t^m representa o estado da m-ésima cadeia no tempo *t*, enquanto Y_t representa a saída de todo o processo (isto é, o valor esperado do RTN multi-nível) no tempo *t* (GHAHRAMANI, 1997; PUGLISI, 2020).

Essa abordagem permite decompor o RTN multi-nível em uma superposição de RTNs de dois níveis: como a saída do FHMM é uma coleção de flutuações de dois níveis, agora é possível recuperar as características distintas de cada flutuação RTN de dois níveis que contribuem para o RTN multi-nível observado (MOR, 2020; PUGLISI, 2014).

Figura 6 - Representação Gráfica de um FHMM.



Fonte: (PUGLISI, 2013b).

Relacionando o conceito de FHMM com o ruído RTN, podemos dizer que o número *K* de cadeias independentes do modelo está diretamente relacionado com o número de armadilhas que contribuem para o sinal observado, ou seja, se existem duas armadilhas

gerando o sinal observado, então o modelo possuirá duas cadeias de Markov que evoluem independentemente uma da outra.

É importante ressaltar que um modelo mais complexo (maior número de estados ocultos no HMM ou maior número de cadeias de Markov no FHMM) resulta em maior tempo de solução. Lamentavelmente, como no HMM, o número de estados ocultos é um parâmetro de entrada para o modelo, então o número de cadeias Markov paralelas (ou seja, o número de armadilhas que contribuem para o RTN observado) deve ser estimado de alguma forma antecipadamente no caso do FHMM. No entanto, esse problema pode ser resolvido alimentando o algoritmo com um número razoavelmente grande de defeitos esperados (embora resultando em uma tarefa mais demorada). Assim, as cadeias relacionadas a armadilhas desnecessárias para coincidir com a entrada RTN serão caracterizadas por amplitudes insignificantes e podem ser facilmente descartadas após a análise. A vantagem do FHMM sobre o HMM é evidente mesmo a esse respeito, pois uma estimativa muito grande do número de estados ocultos no HMM faz com que o algoritmo seja forçado a identificar mais níveis ocultos que o número real. Em vez disso, na abordagem do FHMM, o uso de um grande número de cadeias paralelas não afeta a qualidade do encaixe, tornando o FHMM uma ferramenta autoconsistente e precisa (PUGLISI, 2014).

2.3 Problemas Básicos do HMM

No HMM os estados individuais do modelo são definidos como {1, 2, ..., *N*}, e o estado em determinado tempo *t* é rotulado como q_t. *M* é o número de símbolos de observações distintos por estado, por exemplo, o tamanho do alfabeto discreto. Os símbolos individuais são denotados como $V = \{v_1, v_{2,\ldots,} v_M\}$ (COSTA, 1994; DE OLIVEIRA, 2000).

A distribuição de probabilidade das transições entre os estados, $A = \{a_{ij}\}$, onde a_{ij} é definido pela Equação 11.

$$a_{ij} = P\left(q_{t+1} = j \middle| q_t = i\right), \qquad 1 \leq i, j \leq N \tag{11}$$

No caso em que todos os estados podem ser alcançados a partir de qualquer outro estado, tem-se $a_{ij} > 0$ para todo i, j. Para outros tipos de HMM, onde não existe transição entre determinados estados, podem-se ter $a_{ij} = 0$ para um ou mais pares (i, j) (DE OLIVEIRA, 2000).

A distribuição de probabilidade dos símbolos observáveis, $B = \{b_j(k)\}$, define a distribuição de símbolos no estado j, com j = 1, 2, ..., N (DE OLIVEIRA, 2000). Isto é formalizado através da Equação 12.

$$b_j(k) = P\left(O_t = v_k \middle| q_t = j\right), \qquad 1 \leq k \leq M \tag{12}$$

Já a distribuição de estado inicial $\pi = \{\pi_i\}$ é formalizada através da Equação 13.

$$\pi_i = P(q_1 = i), \qquad 1 \leq i \leq N \tag{13}$$

Existem três questões fundamentais que precisam ser analisadas e resolvidas em um sistema HMM, para que o modelo possa ser utilizado em aplicações do mundo real (JURAFSKY, 2018). Estes problemas são os seguintes:

A primeira questão a ser respondida é qual a probabilidade de que uma determinada sequência de observações $O = \{O_1, O_2, O_3, ..., O_T\}$, tenha sido gerada por um determinado modelo $\lambda = (A, B, \pi)$, ou seja, $P(O|\lambda)$.

O segundo problema busca responder qual é a sequência de estados $Q = \{q_1, q_2, q_3, ..., q_T\}$, que melhor define uma sequência de observações $\{O_1, O_2, O_3, ..., O_T\}$.

A terceira questão é o problema da aprendizagem, ou seja, dada uma sequência de observações $\{O_1, O_2, O_3,..., O_T\}$, como se pode aprender as probabilidades do modelo que gerariam estas observações.

Existem dois algoritmos que são utilizados na resolução destes problemas. O algoritmo *Forward-Backward (Baum-Welch)* é utilizado para solucionar o terceiro problema, da aprendizagem. O algoritmo *Viterbi* é utilizado no segundo problema. E a primeira questão pode ser facilmente respondida apenas através da parte *Forward* do algoritmo *Baum-Welch* (COSTA, 1994; DE OLIVEIRA, 2000).

2.3.1 Solução do Primeiro Problema

O primeiro problema é o da avaliação, isto é, dado um modelo e uma sequência de observações, como calcular a probabilidade que a sequência observada seja produzida por este modelo. A solução deste problema permite escolher um modelo que melhor corresponde às observações (DE OLIVEIRA, 2000).

A fim de calcular a probabilidade de ter uma determinada sequência de observações $O = (O_1, O_2, O_3,..., O_T)$, dado um modelo $\lambda = (A, B, \pi)$, é necessário enumerar todas as possíveis sequências se estados de tamanho *T* (número de observações), através da Equação 14 (DE OLIVEIRA, 2000).

$$P(O|\lambda) = \sum_{q_1,q_2,...,q_T} \pi_{q_1} * b_{q_1}(O_1) * a_{q_1 q_2} * b_{q_2}(O_2) \dots a_{q_{T-1} q_T} * b_{q_T}(O_T) \tag{14}$$

Porém, este não é o método mais prático de realizar este cálculo, pois a Equação 14 envolve uma ordem de $(2T-1)N^T + (N^T - 1)$ cálculos, isto é, para cada $t = 1,2,...,T$, existem *N* possíveis estados que podem ser alcançados, ou seja, existem N^T possíveis

sequências de estados, e para cada uma destas sequências são necessários $2T$ cálculos. Este cálculo se torna computacionalmente inviável, mesmo para pequenos valores de *N* e *T*. Logo, torna-se necessário encontrar uma forma mais eficiente para resolver este problema, e isto é realizado através da utilização do algoritmo *forward-backward* (DE OLIVEIRA, 2000; RAMESH, 2010).

Para que possamos calcular a parte *forward* do algoritmo é necessário introduzir uma variável *alpha* (α). Esta variável é uma função que fornece a probabilidade do sistema estar no tempo *t*, no estado *i*, dada uma sequência de observações que ocorreu antes do estado que está sendo analisado. A Equação 15 mostra a definição desta variável (RABINER, 1989; TENYAKOV, 2014).

$$\alpha_t(i) = P\left(O_1, O_2, O_3, \dots, O_t, q_t = S_i \middle| \lambda\right) \tag{15}$$

Para calcular o *forward,* existem dois casos: o caso base e o caso indutivo. As Equações 16 e 17 representam os dois casos, respectivamente.

$$\alpha_1(i) = \pi_i * b_i(O_1) \qquad 1 \leq i \leq N \tag{16}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) * a(i,j)\right] * b_j(O_{t+1}) \qquad 1 \leq t \leq T-1 \qquad 1 \leq j \leq N \tag{17}$$

O caso base do *forward* é calculado para $t = 1$, e leva em consideração a probabilidade de iniciar no estado *i* e a probabilidade de ver a observação O_1 neste estado.

Já o cálculo do *forward* para qualquer estado que não seja o inicial, precisa considerar todas as formas possíveis de ter-se chegado a este estado. Então, o que o passo indutivo faz é considerar todos os possíveis estados os quais se pode estar no tempo *t*, e sabendo que existe uma sequência de observações anterior a este tempo *t*, utilizar a variável α para calcular a probabilidade de estar no estado S_j no tempo $t + 1$. Logo, se o sistema encontra-se no estado 1 no tempo *t*, é necessário descobrir qual a probabilidade do sistema estar no estado S_j no tempo $t + 1$, dada toda a sequência de observações anterior a este estado. Para isso, é necessário somar todos os estados os quais se poderia estar no tempo *t*, multiplicar pelas possíveis probabilidades de transição para o estado S_j. E em seguida, multiplicar pela probabilidade de ver a observação O_{t+1} no estado S_j (RABINER, 1989; TENYAKOV, 2014).

O passo final deste cálculo é utilizado para resolver o primeiro problema do HMM, e é dado na Equação 18.

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \qquad 1 \leq i \leq N \tag{18}$$

Porém, a fim de resolver o restante das questões necessita-se calcular algumas outras variáveis. A segunda variável necessária é o *beta* (β), também conhecido como variável *backward*.

O β calcula a probabilidade de ter uma dada sequência de observações no futuro, a partir de $t + 1$, sabendo que no tempo *t* o sistema encontra-se no estado S_i, como mostrado na Equação 19. O *backward* é calculado a posteriori, ou seja, depois de se ter toda a sequência,

pois o valor desta variável depende das observações que ainda irão acontecer (RABINER, 1989; TENYAKOV, 2014).

$$\beta_t(i) = P\left(O_{t+1}, O_{t+2}, O_{t+3}, \dots, O_t \middle| q_t = S_i, \lambda\right) \tag{19}$$

Com o objetivo de calcular a variável β percorre-se o mesmo caminho realizado no cálculo da variável α, porém, de trás para frente. Novamente, existem dois casos: o caso base e o caso indutivo, os quais estão demonstrados nas Equações 20 e 21, respectivamente (RABINER, 1989; TENYAKOV, 2014).

$$\beta_t(i) = 1 \tag{20}$$

$$\beta_t(i) = \sum_{j=1}^{N} a(i,j) * b_j(O_{t+1}) * \beta_{t+1}(j) \qquad 1 \leq j \leq N \tag{21}$$

O caso base tem sempre probabilidade igual a 1, visto que não ocorrerão observações no futuro. E o passo indutivo realiza o cálculo da probabilidade do sistema encontrar-se no estado S_i no tempo t, sabendo todas as observações que serão vistas do tempo t em diante (RABINER, 1989; TENYAKOV, 2014).

O cálculo de $\alpha_t(i)$ e $\beta_t(i)$ envolve multiplicação com probabilidades. Todas estas probabilidades possuem um valor menor que 1 (geralmente significantemente menor que 1), e conforme t começa a crescer, cada termo de $\alpha_t(i)$ e $\beta_t(i)$ começa a tender exponencialmente para zero. Para um t suficientemente grande (por exemplo, 100 ou mais) a faixa dinâmica de $\alpha_t(i)$ e $\beta_t(i)$ excederá a faixa de precisão de essencialmente qualquer máquina, ocorrendo o que chamamos de *underflow.* Uma das maneiras encontradas para resolver este problema é

realizar um procedimento conhecido como escalamento (NILSSON, 2005; TATAVARTY, 2011).

O procedimento de escalamento básico multiplica $\alpha_t(i)$ por um coeficiente de escalamento, o qual é dependente apenas do tempo t e independente do estado i. O fator de escalamento para a variável *forward* é denotado por c_t (o escalamento é realizado para cada tempo t, para todos os estados i – $1 \leq i \leq N$). Este mesmo fator será utilizado no escalamento da variável *backward,* $\beta_t(i)$ (NILSSON, 2005).

Para o algoritmo *forward* escalado considera-se a computação da variável *forward,* $\alpha_t(i)$*,* e algumas outras notações. A variável *forward* não escalada é dada por $\alpha_t(i)$, a variável iterada e escalada de $\alpha_t(i)$ é denotada por $\hat{\alpha}_t(i)$, a variável $\hat{\hat{\alpha}}_t(i)$ denota a versão local de $\alpha_t(i)$, ou seja, a versão de $\alpha_t(i)$ antes do escalamento, e c_t representará o coeficiente de escalamento em cada tempo t (KURDTHONGMEE, 2014; NILSSON, 2005). O algoritmo *forward* escalado inclui os seguintes passos:

1. Inicialização

$$Define\ t = 1;$$

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N \tag{22}$$

$$\hat{\hat{\alpha}}_1(i) = \alpha_1(i), \qquad 1 \leq i \leq N \tag{23}$$

$$c_1 = \frac{1}{\sum_{i=1}^{N} \alpha_1(i)} \tag{24}$$

$$\hat{\alpha}_1(i) = c_1 \alpha_1(i) \tag{25}$$

2. Indução

$$\hat{\hat{\alpha}}_t(i) = b_i(O_t) * \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) * a_{j,i} \qquad 1 \leq i \leq N \tag{26}$$

$$c_t = \frac{1}{\sum_{i=1}^{N} \hat{\hat{\alpha}}_t(i)} \tag{27}$$

$$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i) \qquad 1 \leq i \leq N \tag{28}$$

3. Atualização do Tempo

   Define $t = t + 1$;

   Retorna ao passo 2 se $t \leq T$;

   Senão, termina o algoritmo (Passo 4)

4. Terminação

$$log\, P(O|\lambda) = -\sum_{t=1}^{T} log\, c_t \tag{29}$$

A principal diferença entre o algoritmo *forward* padrão e o algoritmo com a etapa de escalamento encontra-se nos passos 2 e 4. E a única mudança real para o procedimento HMM devido ao escalamento, é o procedimento para computar $P(O|\lambda)$. Não se pode apenas somar os termos de $\hat{\alpha}_T(i)$, pois estes já estão escalados. Então, no passo 4 é utilizada a função logarítmica para o cálculo de $P(O|\lambda)$. O logaritmo de $P(O|\lambda)$ é muitas vezes tão útil quanto $P(O|\lambda)$, pois na maioria dos casos esta medida é usada como uma comparação com outras probabilidades (para outros modelos) (NILSSON, 2005).

O algoritmo *backward* escalado pode ser encontrado mais facilmente, já que usará o mesmo fator de escalamento que é usado no algoritmo *forward*. A notação utilizado é similar a notação da variável *forward,* isto é, $\beta_t(i)$ denota a variável *backward* não escalada, $\hat{\beta}_t(i)$ representa a variante escalada e iterada de $\beta_t(i)$, $\hat{\hat{\beta}}_t(i)$ representa a versão local de $\beta_t(i)$ antes do escalamento e c_t representará o coeficiente de escalamento em cada tempo t (KURDTHONGMEE, 2014; NILSSON, 2005). O algoritmo *backward* escalado inclui os seguintes passos:

1. Inicialização

   $Define\ t = T - 1$;

$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \tag{30}$$

$$\hat{\beta}_T(i) = c_T \beta_T(i), \qquad 1 \leq i \leq N \tag{31}$$

2. Indução

$$\hat{\hat{\beta}}_t(i) = \sum_{j=1}^{N} \hat{\beta}_{t+1}(j) * a_{i,j} * b_j(O_{t+1}) \qquad 1 \leq i \leq N \tag{32}$$

$$\hat{\beta}_t(i) = c_t \hat{\hat{\beta}}_t(i) \qquad 1 \leq i \leq N \tag{33}$$

3. Atualização do Tempo

Define $t = t - 1$;

Retorna ao passo 2 se $t > 0$;

Senão, termina o algoritmo.

2.3.2 Solução do Segundo Problema

O segundo problema de um sistema HMM é a questão da busca pela melhor sequência de estados. Este problema procura descobrir a parte escondida do modelo, ou seja, encontrar a sequência de estados correta. A solução geralmente é dada através do algoritmo de *Viterbi* (FORNEY, 1973; LOU, 1995), que procura a melhor sequência de estados $Q = (q_1, q_2, \dots, q_T)$ para uma dada sequência de observações $O = (O_1, O_2, \dots, O_T)$.

Portanto, para encontrar a melhor sequência de estados, para uma dada sequência de observações, define-se a quantidade *delta* (δ) demonstrada na Equação 34 (CHURBANOV, 2008; DE OLIVEIRA, 2000).

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P\left(q_1 q_2 \dots q_{t-1}, q_t = i, O_1 O_2 \dots O_t \mid \lambda\right) \tag{34}$$

A variável $\delta_t(i)$ proporciona o melhor resultado, ou seja, a probabilidade mais alta ao longo de um caminho simples no tempo *t*, o qual leva em consideração todas as *t* primeiras

observações e termina no estado *i*. A parte indutiva do cálculo de δ é mostrada na Equação 35 (CHURBANOV, 2008; DE OLIVEIRA, 2000).

$$\delta_{t+1}(j) = [\max_i \delta_t(i) * a(i,j)] * b_j(O_{t+1}) \tag{35}$$

A fim de recuperar a sequência de estados, é necessário manter os argumentos que maximizam a Equação 35, para cada *t* e *i* através do *array* $\psi_t(j)$. Logo, o procedimento completo para encontrar a melhor sequência de estados é dado através das Equações 36a e 36b que são a etapa de inicialização das variáveis, 37a e 37b as quais realizam a etapa de recursão do algoritmo, 38a e 38b que são a etapa de finalização das variáveis e em seguida a etapa final chamada de *backtracking* dada pela Equação 39 (CHURBANOV, 2008; DE OLIVEIRA, 2000).

$$\delta_t(i) = \pi_i * b_i(O_1) \qquad 1 \leq i \leq N \tag{36a}$$

$$\psi_1(i) = 0 \tag{36b}$$

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) * a(i,j)] * b_j(O_t) \qquad 2 \leq t \leq T \quad 1 \leq j \leq N \tag{37a}$$

$$\psi_t(j) = arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) * a(i,j)] \qquad 2 \leq t \leq T \quad 1 \leq j \leq N \tag{37b}$$

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \tag{38a}$$

$$q_T{}^* = arg \max_{1 \le i \le N} [\delta_T(i)] \tag{38b}$$

$$q_t{}^* = \psi_{t+1}(q_{t+1}{}^*) \qquad t = T-1, T-2, \dots, 1 \tag{39}$$

Com exceção da etapa de *backtracking*, o algoritmo *Viterbi* e o procedimento *forward* possuem basicamente a mesma implementação. A única diferença entre eles é que o somatório do procedimento *forward* é trocado pela maximização no algoritmo *Viterbi* (CHURBANOV, 2008; DE OLIVEIRA, 2000).

2.3.3 Solução do Terceiro Problema

O problema da aprendizagem é a terceira questão a ser resolvida de um HMM, e também é a mais difícil. Para dar início ao entendimento desta questão é necessário introduzir a variável *gamma* (γ). Esta variável fornece a probabilidade do sistema estar em um dado estado S_i, no tempo *t*, dada uma sequência de observações. A variável γ é definida pela Equação 40 (CHURBANOV, 2008; RABINER, 1989).

$$\gamma_t(i) = P\left(q_t = S_i \middle| O, \lambda\right) \tag{40}$$

Para realizar o cálculo de γ utiliza-se o α (*forward*) e o β (*backward*), calculados anteriormente. Assim, a Equação 40 pode ser reescrita como a Equação 41.

$$\gamma_t(i) = \frac{\alpha_t(i) * \beta_t(i)}{P(O|\lambda)} \qquad 1 \leq i \leq N \tag{41}$$

O α fornece a probabilidade da primeira parte do caminho, ou seja, qual a probabilidade do sistema estar em um determinado estado, no tempo *t*, sabendo que a primeira parte da sequência de observações até o estado em questão já foi vista. Enquanto o β fornece a probabilidade do sistema encontrar-se neste mesmo estado, neste mesmo tempo *t*, sabendo que o restante das observações ainda será visto no futuro (CHURBANOV, 2008; RABINER, 1989).

Ao fazer o produto de α e β, leva-se em consideração todo o contexto da sequência de observações. Apesar de somente o produto destas variáveis permitir a comparação de diferentes estados para a mesma observação, a fim de saber qual é o estado mais provável, ainda é preciso dividir pelo denominador P(O|λ). Este denominador é a resposta do primeiro problema, e ele funciona aqui como um normalizador para o numerador, ou seja, assim pode-se garantir que a soma de todos os estados de $\gamma_t(i)$ seja 1, garantindo então, que todos os $\gamma_t(i)$ sejam probabilidades (CHURBANOV, 2008; RABINER, 1989).

Por último, é preciso calcular a variável ξ, a qual é uma função que calcula a probabilidade de que no tempo *t*, o sistema estará no estado S_i, e no tempo $t + 1$, estará no estado S_j, dado o conjunto de observações e o modelo λ. A definição desta variável é dada pela Equação 42 (CHURBANOV, 2008; RABINER, 1989).

$$\xi_t(i,j) = \frac{\alpha_t(i) * a(i,j) * b_j(O_{t+1}) * \beta_{t+1}(j)}{P(O|\lambda)} \qquad 1 \leq t \leq T-1 \qquad 1 \leq i,j \leq N \tag{42}$$

O α fornece a probabilidade do sistema se encontrar no estado S_i, no tempo *t*, dada a sequência de observações que precede o estado em questão. Em seguida, tem-se uma probabilidade de transição do estado S_i para o estado S_j, sendo que S_j é o estado que o sistema está no tempo $t+1$. Depois, tem-se a probabilidade de emissão, que corresponde a observação que será visualizada no estado S_j. Por ultimo, ainda é necessário multiplicar pelo *backward*, para considerar o restante das observações que irão ocorrer no futuro (CHURBANOV, 2008; RABINER, 1989).

Novamente, o denominador é utilizado como um fator de normalização para o numerador, garantindo que os valores calculados serão probabilidades e sua soma será 1.

Agora, com todas as variáveis definidas, pode-se realizar a resolução do terceiro problema do HMM. Para a resolução deste problema, é utilizado o algoritmo *Baum-Welch*, o qual é essencialmente um algoritmo *Expectation-Maximization* (EM) (MOON, 1996) aplicado a um HMM. Este algoritmo fornece as probabilidades de transição mais prováveis, e o conjunto de probabilidades de emissão mais provável, considerando apenas os estados observados do modelo.

O algoritmo *Baum-Welch* resolve a seguinte questão: dado um modelo HMM λ e uma sequência de observações, como deve-se ajustar os parâmetros do modelo λ = (A, B, π) de modo a maximizar a P(O| λ). Este algoritmo é resolvido em duas etapas (RABINER, 1989):

- Calcular $\alpha, \beta, \gamma e \xi$
- Realizar a atualização do modelo λ = (A, B, π)

Para atualizar o modelo λ, utiliza-se as Equações 43, 44 e 45.

$$\bar{\pi}_i = \gamma_1(i) \qquad 1 \leq i \leq N \tag{43}$$

$$\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad 1 \leq i,j \leq N \tag{44}$$

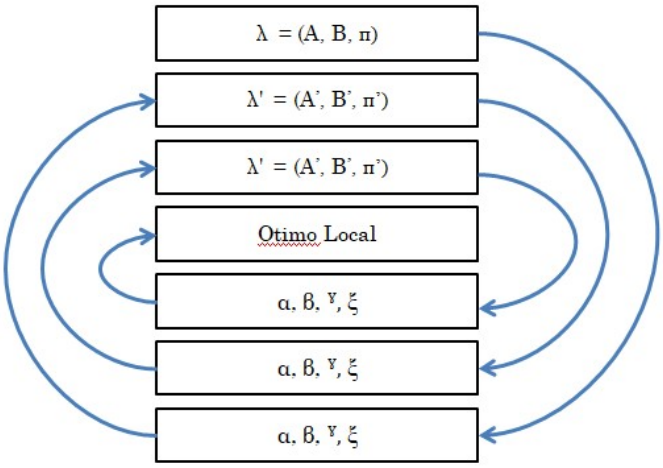
$$\bar{b}_j(v_k) = \frac{\sum_{t=1 \; em \; que \; o \; símbolo = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \qquad 1 \leq j \leq N \tag{45}$$

Então, para atualizar o modelo faz-se uso das variáveis já calculadas anteriormente. A variável $\gamma_t(i)$ é a probabilidade de que o sistema encontre-se no estado S_i, no tempo *t*. Então, realizando a soma de todos os tempos *t*'s em que o estado do sistema é S_i, tem-se um valor que pode ser tratado como o número esperado de vezes que o estado S_i é visitado. E $\xi_t(i,j)$ é a probabilidade de ocorrer uma transição do estado S_i para o estado S_j, no tempo *t*. Logo, fazendo a soma de todos os tempos *t*, tem-se um valor que diz qual é o número esperado de vezes que esta transição acontece (RABINER, 1989).

Portanto, para atualizar a matriz de transição faz-se a divisão do número esperado de vezes em que ocorre a transição de S_i para S_j pelo número esperado de vezes em que ocorre uma transição a partir do estado S_i. Da mesma forma, para atualizar a matriz de emissão, divide-se o número esperado de vezes em que o sistema encontra-se no estado S_j (e observa-se o símbolo v_k) pelo número esperado de vezes em que o sistema esteve nesse estado S_j (RABINER, 1989).

Logo, pode-se resumir o funcionamento do algoritmo *Baum-Welch* da seguinte maneira: dado um modelo λ = (A, B, π) e uma sequência de observações O, podem-se produzir as variáveis *α, β, γ* e *ξ*. E dada estas variáveis, consegue-se calcular um novo modelo $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. Como este é um algoritmo iterativo, o qual converge para um valor ótimo local, então é necessário executá-lo várias vezes para que se consiga a melhor resposta. Este processo está simplificado na Figura 7 (RABINER, 1989).

Figura 7 - Representação da Iteração realizada no Algoritmo Baum-Welch.



Fonte: Elaborado pela autora.

As Equações 41 e 42 são utilizadas quando se está trabalhando com variáveis *forward* e *backward* não escaladas. Se variáveis escaladas forem utilizadas, a probabilidade $\xi_t(i,j)$ será encontrada como na Equação 46 (KURDTHONGMEE, 2014; NILSSON, 2005).

$$\xi_t(i,j) = \frac{\hat{\alpha}_t(i) * a(i,j) * b_j(O_{t+1}) * \hat{\beta}_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\alpha}_t(i) * a(i,j) * b_j(O_{t+1}) * \hat{\beta}_{t+1}(j)} \tag{46}$$

E a variável $\gamma_t(i)$ escalada é dada pela Equação 47.

$$\gamma_t(i) = \frac{\hat{\alpha}_t(i) * \hat{\beta}_t(i)}{\sum_{i=1}^{N} \hat{\alpha}_t(i) * \hat{\beta}_t(i)} \tag{47}$$

3 MÉTODOS UTILIZADOS NA EXTRAÇÃO DOS PARÂMETROS DE UM RTN

Neste capítulo serão apresentados dois métodos implementados para a extração de parâmetros de um sinal RTN de dois níveis. O primeiro método, o qual foi desenvolvido neste trabalho baseia-se no algoritmo *Baum-Welch.* E um segundo método, que foi desenvolvido com base na discretização de medidas. Neste trabalho, estes métodos serão denominados, respectivamente, método A e método B. Além disso, será apresentado o algoritmo utilizado na geração de um sinal RTN sintético. A geração de RTN sintético permite a validação e comparação dos métodos implementados, visto que os parâmetros do RTN gerado são conhecidos.

Os algoritmos apresentados aqui foram desenvolvidos no software R, em linguagem de programação R (COGHLAN, 2011). O R é um software livre utilizado para computação estatística e construção de gráficos que pode ser baixado e distribuído gratuitamente com a licença GNU. Além disso, o R é uma ferramenta utilizada para armazenar e manipular dados, realizar cálculos, realizar testes estatísticos, análises exploratórias e produzir gráficos (LANDEIRO, 2011; RITTER, 2019).

3.1 Algoritmo de Geração do RTN Sintético

O algoritmo implementado tem como objetivo gerar uma série temporal de um sinal RTN sintético de dois níveis, ou seja, esta série temporal apresentará dois valores observáveis distintos. Estes dados serão utilizados posteriormente, para realizar os testes de funcionamento dos algoritmos de extração de parâmetros, os quais foram desenvolvidos com o objetivo de extrair do sinal RTN sintético gerado as informações necessárias que definem este sinal, ou seja, os valores dos tempos de captura (τ_c) e emissão (τ_e), e a amplitude (ΔI) do sinal RTN.

Foram implementadas duas versões do mesmo algoritmo. Na primeira versão, o algoritmo gera um sinal RTN limpo, sem adição de ruído. E na segunda versão, um ruído térmico é adicionado ao sinal.

O ruído térmico, também conhecido como ruído Johnson-Nyquist, é o ruído gerado pela agitação térmica de cargas no interior de um condutor elétrico em equilíbrio, e é independente da corrente aplicada. Este ruído é aproximadamente um ruído branco, ou seja, a sua densidade espectral de potência é aproximadamente constante ao longo do espectro de frequências. Devido a isso, pode-se dizer que o sinal é praticamente Gaussiano (BOTH, 2017).

A primeira versão do algoritmo de geração do sinal RTN é apresentada através do pseudocódigo Algoritmo 1. Em um primeiro momento são definidas as variáveis *final_time*, *Nt* e *AmplAvg* que são valores numéricos. A variável *Nt* representa o número de armadilhas existentes, que no caso deste trabalho será definido como 1, pois o sinal RTN gerado será um sinal de dois níveis. A variável *AmplAvg* irá receber um valor que representará a amplitude deste sinal. Já a variável *final_time* representa o número de amostras que o sinal gerado terá, ou seja, será o tamanho total da amostra de dados que será gerada.

Posteriormente, têm-se as variáveis *RTS*, *Ampl*, *Tc*, *Te* e *State*, as quais são inicializadas como vetores preenchidos com o valor zero. As variáveis *Ampl*, *Tc*, *Te* e *State* serão vetores de tamanho um, pois este sinal é gerado por apenas uma armadilha. Logo, a variável *Ampl* receberá o valor estipulado em *AmplAvg*, as variáveis *Tc* e *Te* receberão os valores definidos para tempo de captura e tempo de emissão, respectivamente, e a variável *State* irá receber um valor que representa o estado inicial da armadilha, ou seja, o estado em que o sistema estará no tempo *t*=1. Por fim, a variável *RTS* receberá os valores gerados deste sinal a cada tempo t.

Algoritmo 1 – Gera o sinal RTN
1: VAR

```
2:      final_time, Nt, AmplAvg: numérico
3:      RTS, Ampl, Tc, Te, State: vetor
4:
5:   PARA (i = 1 até Nt ) FAÇA
6:          Gera um número aleatório entre 0 e 1
7:          SE (número aleatório > Tc[i] / Tc[i] + Te[i])
8:                State ← 0
9:           FimSE
10:          SENÃO
11:               State ← 1
12:         FimSENÃO
13:  FimPARA
14:
15:  PARA (t = 1 até final_time-1 ) FAÇA
16:          PARA (i = 1 até Nt ) FAÇA
17:                   tmp ← número aleatório entra 0 e 1
18:                SE (State[i]==1)
19:                    RTS[t]=RTS[t]+(1-Tc[i]/(Tc[i]+Te[i]))*Ampl[i]
20:                    SE ((1/Tc[i])>tmp[i])
21:                        State[i]=0
22:                    FimSE
23:                FimSE
24:                SENÃO
25:                    RTS[t]=RTS[t]+(-Tc[i]/(Tc[i]+Te[i]))*Ampl[i]
26:                    SE ((1/Te[i])>tmp[i])
27:                        State[i]=1
28:                    FimSE
29:                FimSENÃO
30:          FimPARA
31:  FimPARA
32:
33:  Plota RTS
34:  Grava RTS em um arquivo .csv
```

Cada armadilha tem seu estado inicial definido pela análise da probabilidade de estado ocupado $p_0(t)$ e um número aleatório r entre 0 e 1, gerado no instante da análise. Por definição, tem-se que a armadilha iniciará no estado ocupado, representado pelo valor 1, se $p_0(t) \geq r$, ou iniciará no estado desocupado, representado pelo valor 0, caso $p_0(t) < r$ (MELOS, 2018). Este processo está representado entre as linhas 5 e 13 do pseudocódigo Algoritmo 1, onde é gerado um valor aleatório entre 0 e 1 e em seguida é feita a verificação se este valor gerado é maior do que o resultado da Equação 48, a qual fornece o valor de $p_0(t)$. Caso a afirmação seja verdadeira, a variável *State* receberá o valor 0, caso contrário, esta variável receberá o valor 1.

$$p_0(t) = 1 - \frac{\tau_c}{\tau_c + \tau_e} \tag{48}$$

Posteriormente, é necessário realizar o cálculo do sinal *RTS* em cada instante de tempo *t*. Uma vez que a média deve ser zero, a variável *Ampl* é multiplicada por $1 - p_1(t)$ ou por $-1 * p_1(t)$, sendo que $p_1(t)$ é dado pela Equação 49, e representa a probabilidade de estado desocupado. Logo, se o estado inicial for igual a 1 (armadilha ocupada), a variável *Ampl* é multiplicada pela probabilidade de estado ocupado, $p_0(t)$, e se o estado inicial for igual a 0 (armadilha desocupada), a variável *Ampl* é multiplicada por $-1 * p_1(t)$, onde $p_1(t)$ representa a probabilidade de estado desocupado. Este processo é realizado entre as linhas 15 e 31 do pseudocódigo Algoritmo 1.

$$p_1(t) = \frac{\tau_c}{\tau_c + \tau_e} \tag{49}$$

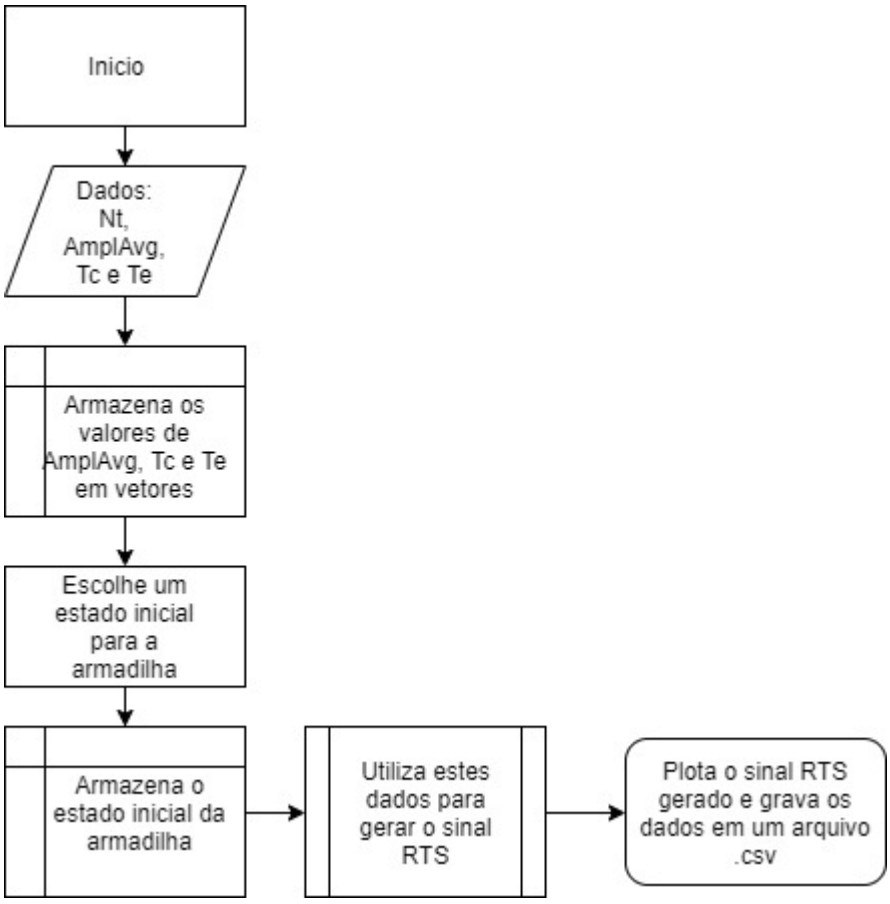
Em todas as iterações do cálculo de *RTS*, a análise do estado da armadilha é refeito, comparando a probabilidade de troca de estado $p(t)$ com um número aleatório r entre 0 e 1 gerado no instante da análise. Similar à definição do estado inicial, pode-se dizer que: se $p(t) \geq r$, ocorrerá à troca de estado, e se $p(t) < r$, o estado permanece o mesmo. Onde $p(t)$ é dado pela Equação 50 se o estado for ocupado, e pela Equação 51 se o estado for desocupado.

$$p(t) = \frac{dt}{\tau_e} \tag{50}$$

$$p(t) = \frac{dt}{\tau_c} \tag{51}$$

Considera-se dt o passo de análise com o intervalo mínimo de duração, neste trabalho, definido em 1 segundo. Por fim, os dados gerados do sinal *RTS* em cada instante de tempo *t*, são gravados em um arquivo de dados com extensão .csv para serem utilizados nos métodos de extração de parâmetros desenvolvidos. Um fluxograma deste algoritmo é mostrado na Figura 8.

Figura 8 - Fluxograma do Algoritmo 1.



Fonte: Elaborado pela autora.

Já a segunda versão deste algoritmo é apresentada através do pseudocódigo Algoritmo 2. Nesta versão o algoritmo funciona da mesma forma que a anterior, porém agora, tem-se a adição do ruído térmico ao sinal. Assim, é preciso definir mais algumas variáveis como *TN*

que será uma matriz que irá guardar os valores de ruído gerado, e também a variável *RTS_TN*, a qual irá armazenar os valores do sinal *RTS* com a adição do sinal de ruído.

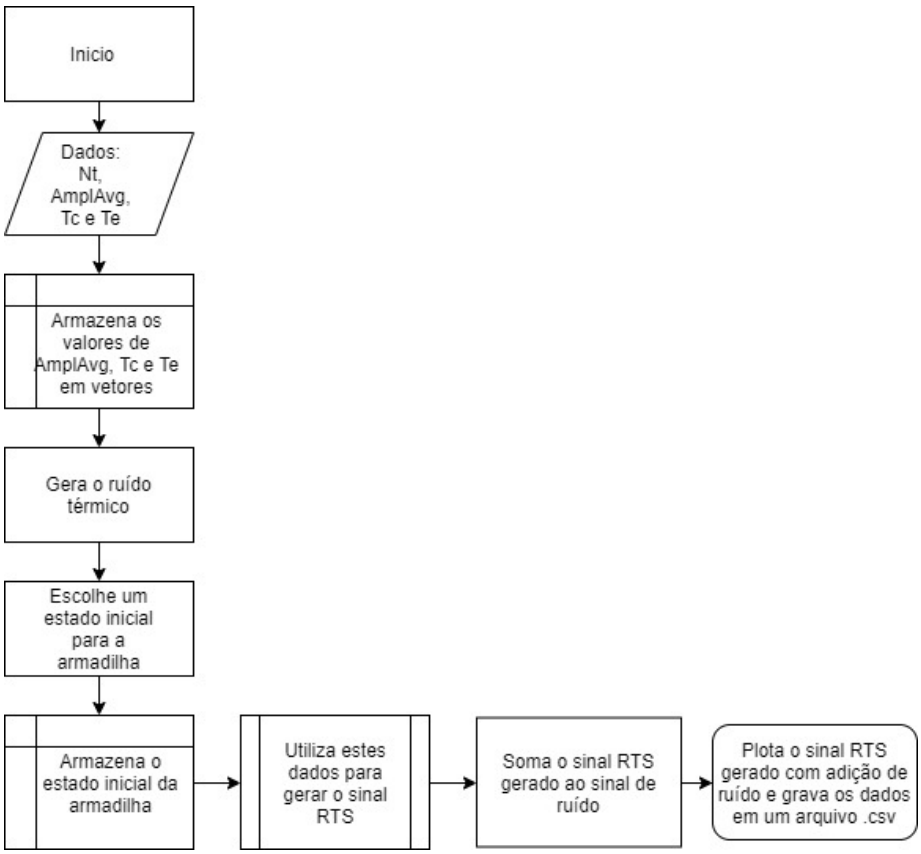
Algoritmo 2 – Gera o sinal RTN com adição de um ruído térmico

```
1:  VAR
2:     final_time, Ndev, Nt, AmplAvg: numérico
3:     RTS, RTS_TN, Ampl, Tc, Te, State: vetor
4:     TN: matriz
5:
6:  PARA (j = 1 até Ndev ) FAÇA
7:         PARA (t = 1 até final_time-1) FAÇA
8:                 TN[j, t] = rnorm(1, mean = 0, sd = 0.004 a 4)
9:         FimPARA
10: FimPARA
11:
12: PARA (i = 1 até Nt ) FAÇA
13:         Gera um número aleatório entre 0 e 1
14:         SE (número aleatório > Tc[i] / Tc[i] + Te[i])
15:               State ← 0
16:         FimSE
17:         SENÃO
18:              State ← 1
19:        FimSENÃO
20:  FimPARA
21:
22:  PARA (t = 1 até final_time-1 ) FAÇA
23:          PARA (i = 1 até Nt ) FAÇA
24:                  tmp ← número aleatório entra 0 e 1
25:                SE (State[i]==1)
26:                    RTS[t]=RTS[t]+(1-Tc[i]/(Tc[i]+Te[i]))*Ampl[i]
27:                    SE ((1/Tc[i])>tmp[i])
28:                        State[i]=0
29:                    FimSE
30:                FimSE
31:                SENÃO
32:                    RTS[t]=RTS[t]+(-Tc[i]/(Tc[i]+Te[i]))*Ampl[i]
33:                    SE ((1/Te[i])>tmp[i])
34:                        State[i]=1
35:                    FimSE
36:                FimSENÃO
37:           FimPARA
38:  FimPARA
39:
40:  PARA (t = 1 até final_time-1 ) FAÇA
41:          RTS_TN[t] = RTS[t] + TN[t]
42:  FimPARA
43:
44:  Plota RTS com ruído
45:  Grava RTS com adição do ruído em um arquivo .csv
```

O ruído térmico é gerado entre as linhas 6 e 10 do pseudocódigo Algoritmo 2, e é dado por uma distribuição normal (Gaussiana). É importante ressaltar que uma distribuição normal possui dois parâmetros, a média (μ), ou seja, onde a distribuição está centralizada, e a variância ($\sigma^2 > 0$) que descreve o seu grau de dispersão. O parâmetro utilizado no algoritmo é o desvio padrão (σ, a raiz quadrada da variância), neste trabalho identificado por *sd*. Cabe salientar que como qualquer outro modelo, dependendo dos parâmetros, têm-se diferentes distribuições normais.

Depois de gerado o ruído térmico é necessário somar este ruído ao sinal *RTS*, e isso é realizado entre as linhas 40 e 42 do pseudocódigo Algoritmo 2. Um fluxograma deste algoritmo é dado na Figura 9.

Figura 9 - Fluxograma do Algoritmo 2.



Fonte: Elaborado pela autora.

3.2 Método de Extração de Parâmetros Baseado no Algoritmo Baum-Welch (Método A)

O algoritmo implementado tem como objetivo extrair os parâmetros de um RTN (τ_c, τ_e e ΔI), com base em apenas uma sequência de observações (série temporal), a qual é produzida através do algoritmo de geração do RTN sintético.

A implementação deste algoritmo se baseia no algoritmo *Baum-Welch* (RABINER, 1989), que é um caso especial do algoritmo de Expectativa-Maximização (EM) (DEMPSTER, 1977), usado para encontrar os parâmetros desconhecidos de um Modelo de Markov Escondido (HMM).

Em estatística, o algoritmo de Expectativa-Maximização (EM) é um método iterativo para estimar parâmetros em modelos estatísticos, quando o modelo depende de variáveis não observadas, ou seja, escondidas. A iteração EM alterna entre executar uma etapa de expectativa (E), e uma de maximização (M). A etapa de expectativa cria uma função para a expectativa da verossimilhança logarítmica usando a estimativa atual para os parâmetros. A etapa de maximização (M) calcula parâmetros para maximizar a verossimilhança logarítmica encontrada na etapa E. Essas estimativas de parâmetro são usadas para determinar a distribuição das variáveis escondidas na próxima etapa E, e o algoritmo se repete várias vezes (por isso é chamado iterativo) (DE ESTATÍSTICA, 2011).

O algoritmo *Baum-Welch* utiliza o algoritmo *forward-backward* na etapa de Expectativa (E), a fim de calcular as estatísticas desta etapa. Isto é, através do algoritmo *forward-backward* é possível fazer uma estimativa de probabilidade máxima dos parâmetros HMM, dada a sequência de observações.

Logo, o algoritmo *Baum-Welch* fornece as probabilidades de transição dos estados ocultos mais prováveis, e o conjunto de probabilidades de emissão dos estados observados mais provável, considerando apenas a sequência de observações visíveis do modelo.

Um fluxograma completo do algoritmo para extrair os parâmetros de um Modelo de Markov Escondido pode ser visualizado na Figura 10. Neste fluxograma é possível ter uma visão geral das etapas executadas, as quais serão descritas separadamente.

O algoritmo inicia lendo o arquivo de observações, que foram geradas através do código de geração do RTN sintético. Além da sequência de observações, no passo inicial também são computadas as estimativas de probabilidades iniciais do nosso modelo, denominado $\lambda = (A, B, \pi)$.

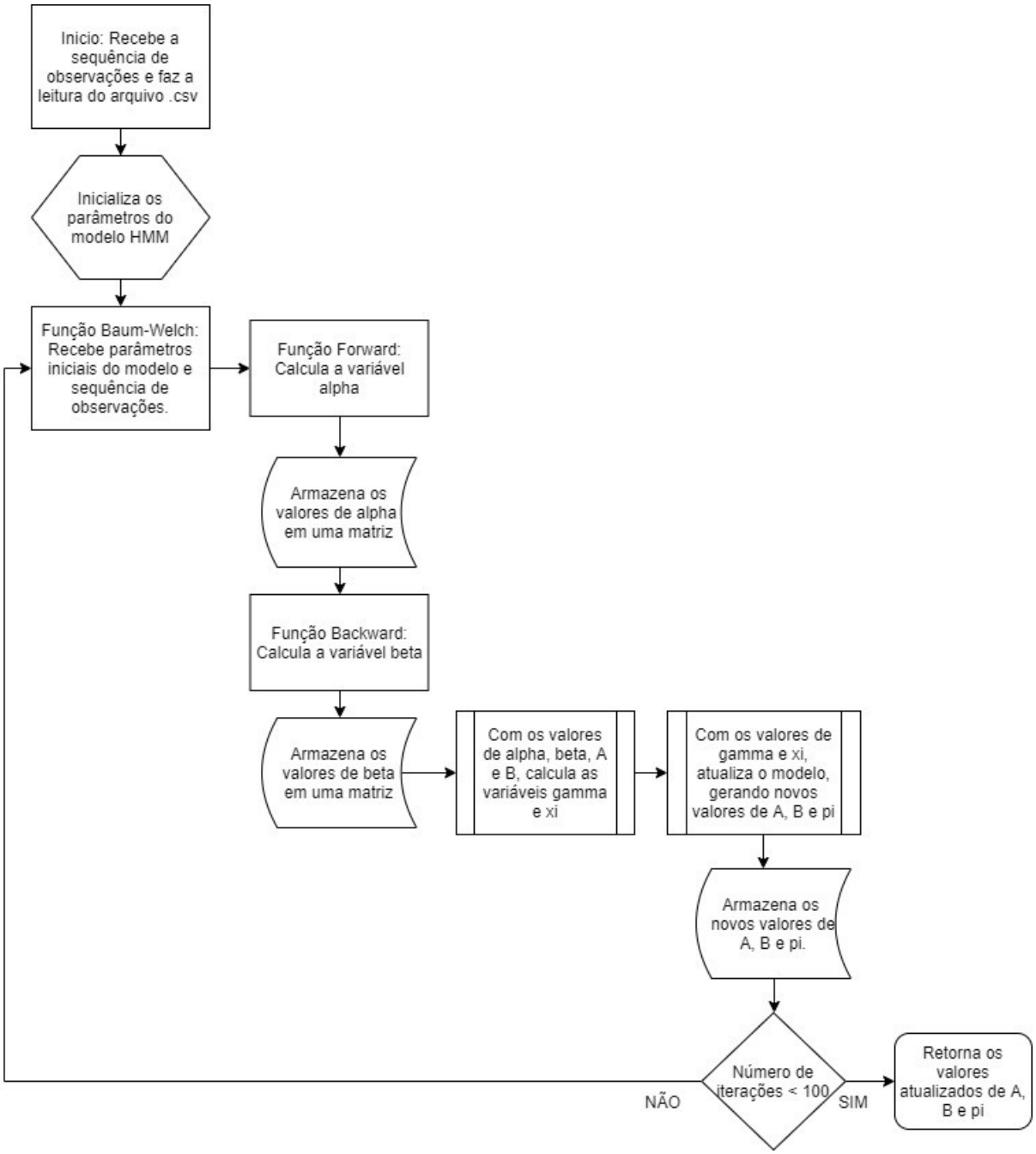
Antes que a reestimação de parâmetros de *Baum-Welch* possa ser aplicada é importante que se obtenha bons parâmetros iniciais, para que assim a reestimação possa levar a um máximo global ou o mais próximo disso possível. Uma escolha adequada para π e A é a distribuição uniforme quando um modelo ergódico é usado. Um modelo ergódico é aquele no qual todo estado pode ser alcançado a partir de qualquer outro estado do modelo, ou seja, é um modelo completamente conectado. Um modelo ergódico possui a propriedade de que $0 < a_{i,j} < 1$ (NILSSON, 2005).

Em seguida, estes parâmetros são utilizados na etapa de Expectativa (E), pelas funções *forward* e *backward*, com o objetivo de computar a expectativa de quantas vezes cada transição/emissão foi utilizada. Em seguida, com estes valores armazenados, poderemos estimar variáveis latentes (γ e ξ), ou seja, variáveis que não são diretamente observadas.

Logo, com o resultado da etapa de Expectativa (E), é possível passar para a etapa de Maximização (M), onde o modelo é atualizado. Ou seja, baseado nas estimativas calculadas anteriormente, é possível fazer uma re-estimativa das probabilidades dos parâmetros do modelo $\lambda = (A, B, \pi)$.

O algoritmo *Baum-Welch* é repetido várias vezes, a fim de encontrar um valor máximo local para a estimativa das probabilidades de transição (A) e emissão (B).

Figura 10 - Fluxograma Geral do Método de Extração de Parâmetros.



Fonte: Elaborado pela autora.

A fim de esclarecer o funcionamento do algoritmo para extração dos parâmetros do RTN, será feita uma descrição detalhada de cada etapa do código.

No pseudocódigo Algoritmo 3, é possível verificar a etapa de inicialização do algoritmo. Nesta etapa, o arquivo "Data.csv" é lido, e seus valores são guardados em um vetor

denominado *O*. Ou seja, no vetor *O* são armazenados os valores RTS (RTN no domínio do tempo), a cada passo de tempo, que neste caso é $\Delta t = 1$.

Neste trabalho, apenas uma armadilha está sendo considerada na geração dos dados do RTN sintético. Logo, as variáveis *N* e *M* recebem o valor 2, já que existem apenas dois estados escondidos e dois valores observáveis distintos.

Em seguida, os dois valores observáveis são separados, e então o vetor das observações (*O*) recebe o valor 1 onde tem-se o valor máximo do RTS, e recebe o valor 2 onde tem-se o valor mínimo do RTS. Esta etapa é realizada para que possamos trabalhar com valores discretos e não valores flutuantes durante o restante do algoritmo.

A amplitude do sinal pode ser calculada através da separação dos valores observáveis. Para isso, a função "*mean*" é utilizada, ou seja, é feita a média dos valores quando a armadilha está em um dos estados e a média dos valores quando esta armadilha encontra-se no segundo estado. Com isso, pode-se fazer a diferença entre os dois valores encontrados, e assim tem-se a amplitude do sinal RTN.

Por fim, é realizada uma estimativa inicial das probabilidades dos parâmetros *A*, *B* e π. Estes parâmetros serão atualizados através da função *Baum-Welch*, que será explicada em seguida.

Algoritmo 3 – Inicialização

```
1:  data ← arquivo de dados Data.csv
2:
3:  VAR
4:        O: vetor
5:        M,N: numérico
6:        A, B: matriz
7:        π: vetor
8:
9:  Amplitude ← a diferença entre os valores observáveis distintos
```

Na função *Baum-Welch* são realizados todos os cálculos necessários para que se possa fazer a re-estimativa dos parâmetros do modelo (*A*, *B* e *π*). O pseudocódigo Algoritmo 4 mostra a implementação desta função.

Algoritmo 4 – Função Baum-Welch

```
1:  FUNÇÃO BaumWelch(v, A, B, initial_distribution, n.iter = 100)
2:
3:    PARA (i = 1 até n.iter) FAÇA
4:          T, N, K: numérico
5:          alpha, beta, Scoef: vetor
6:          xi: array
7:          gamma: matriz
8:
9:          PARA (t = 1 até T) FAÇA
10:               denominatorgamma = (alpha[t,1]*beta[t,1])+(alpha[t,2]*beta[t,2])
11:               PARA (s = 1 até N) FAÇA
12:                     numeratorgamma = alpha[t,s] * beta[t,s]
13:                     gamma[t,s] <- numeratorgamma/denominatorgamma
14:               FimPARA
15:         FimPARA
16:
17:         PARA (t = 1 até T-1) FAÇA
18:               denominator = ((alpha[t,] %*% a) * b[,v[t+1]]) %*% matrix(beta[t+1,])
19:               PARA (s = 1 até N) FAÇA
20:                     numerator = alpha[t,s] * a[s,] * b[,v[t+1]] * beta[t+1,]
21:                     xi[s,,t] = numerator/as.vector(denominator)
22:               FimPARA
23:         FimPARA
24:
25:         initial_distribution[1] = gamma[1,1]
26:         initial_distribution[2] = gamma[1,2]
28:
29:         gammasum <- colSums(gamma[1:(T-1),])
30:
31:         xi.all.t = rowSums(xi, dims = 2)
32:         a = xi.all.t/gammasum
33:
34:         PARA (s = 1 até K) FAÇA
35:               b[,s] = colSums(gamma[which(v==s),])
36:         FimPARA
37:         b = b/colSums(gamma)
38:   FimPARA
39:   Retorna os valores da função
40: FimFUNÇÃO
```

Na linha 1 do pseudocódigo 4 a função *Baum-Welch* recebe os parâmetros iniciais do modelo e também os valores do vetor *O*, que são as observações. Além disso, a função

também recebe um valor indicando o número de vezes que ela deve repetir o processo, para que assim se consiga chegar a valores mais precisos ao final de toda etapa de atualização.

Em seguida, nas linhas 3 a 7 são inicializadas algumas variáveis, que serão utilizadas para definir o tamanho das matrizes utilizadas nos cálculos. As funções *forward* e *backward* serão detalhadas separadamente. A função *forward* calcula a variável α e a função *backward* calcula a variável β.

A variável *alpha* (*forward*) fornece a probabilidade de estar em um determinado tempo *t*, em um dado estado *i*, dada uma sequência de observações que ocorreu antes. Calcula-se o *forward* para todos os estados correspondentes a uma dada observação da sequência de dados.

Já a variável *beta* (*backward*) calcula a probabilidade de se ter uma dada sequência de observações no futuro, a partir de $t + 1$, sabendo que no tempo *t* o sistema encontra-se no estado *i*. Portanto, o *backward* é calculado a posteriori, ou seja, depois de se conhecer toda a sequência de observações, pois seu valor depende das observações que ainda irão acontecer.

Por último, são inicializadas as variáveis *gamma* e *xi*. A variável *gamma* será uma matriz de dimensão (*T*x*N*), onde *T* é o tamanho do vetor de observações e *N* é o número de linhas do parâmetro *A* do modelo. Logo, *T* representa cada passo de tempo *t* na matriz *gamma* e *N* representa cada um dos estados escondidos existentes.

A variável *gamma* é calculada entre as linhas 9 e 15 da função *Baum-Welch*. Essa variável fornece a probabilidade de estar em um dado estado *i*, no tempo *t*, dada a sequência de observações. Para este cálculo utilizam-se as variáveis *alpha* e *beta*, pois a primeira fornece a probabilidade da primeira parte do caminho (ou seja, a probabilidade de estar em um determinado estado, no tempo *t*, sabendo que já foram vistas as observações anteriores a este tempo *t* em análise), e a segunda fornece a probabilidade da segunda parte do caminho (ou seja, a probabilidade de estar neste mesmo estado, no mesmo tempo *t*, sabendo o restante

da sequência de observações). Portanto, fazendo o produto de *alpha* e *beta*, leva-se em consideração todo o contexto da sequência de observações.

Porém, apesar de somente o produto de *alpha* e *beta* permitir a comparação entre diferentes estados para a mesma observação, ainda é preciso dividir os valores por um denominador, o qual normaliza o numerador e garante que a soma de todos os *gamma* em um determinado passo de tempo *t*, seja igual a 1. Isso garante que todos os elementos da matriz *gamma* sejam probabilidades. Este denominador normalizador é chamado de P(O|λ), e é a resposta ao problema de avaliação do HMM, ou seja, a resposta da função *forward*.

Então, como dito anteriormente, *gamma* é uma matriz de dimensão (*T*x*N*), ou seja, cada linha da matriz *gamma* fornece um passo de tempo *t*, e cada coluna representa os estados distintos existentes. Por exemplo, o elemento (1,1) desta matriz diz qual a probabilidade de que no tempo *t*=1, o sistema estará no estado *i*=1. Assim como, o elemento (1,2) da matriz fornecerá a probabilidade de se estar no tempo *t*=1 e no estado *i*=2.

O próximo passo da função *Baum-Welch* é realizado entre as linhas 17 e 23 do pseudocódigo Algoritmo 4. Essa etapa faz o cálculo da variável *xi*, a qual fornece a probabilidade de estar em um tempo *t*, no estado *i*, e em um tempo $t + 1$, estar no estado *j*. Logo, pode-se dizer que a variável *xi* fornece as probabilidades de transição entre um estado e outro, em um determinado tempo *t*. Assim, existe uma variável *xi* para cada passo de tempo *t*. Esta variável é uma matriz de dimensão (*N*x*N*), onde cada elemento da matriz fornece a probabilidade de transicionar entre o estado *i* e o estado *j*, para aquele determinado passo de tempo *t*.

No cálculo de *xi*, que pode ser visualizado na Equação 46, tem-se *alpha* que fornece a probabilidade de estar no estado *i*, no tempo *t*, dada a sequência de observações que precede o estado em questão. Em seguida, tem-se uma probabilidade de transição do estado *i* para o estado *j*, sendo que o estado *j* é o estado em que o sistema estará em $t + 1$. Depois, ainda

multiplica-se por uma probabilidade de emissão, a qual corresponde à observação que será vista em *j*. Finalmente, é feita a multiplicação pela variável *beta*, para considerar o restante das observações que irão ocorrer no futuro. Novamente, o denominador é utilizado como um fator de normalização para o numerador, garantindo que os valores calculados serão probabilidades e sua soma será 1.

O próximo passo da função *Baum-Welch* é a fase de atualização dos parâmetros do modelo (λ = A, B e π). Para atualizar o modelo faz-se uso das Equações 43, 44 e 45.

A atualização de *π*, vista nas linhas 25 e 26, é dada pela variável *gamma* no tempo *t*=1, ou seja, é dada pela frequência esperada de se estar no estado *i*, no tempo *t*=1.

A atualização de *A* é realizada entre as linhas 29 e 32. Esta atualização é realizada através da divisão da linha 32, onde o numerador é dado pelo número esperado de vezes em que ocorreram transições do estado *i* para o estado *j*, e o denominador é dado pelo número esperado de vezes em que ocorreram transições a partir do estado *i*.

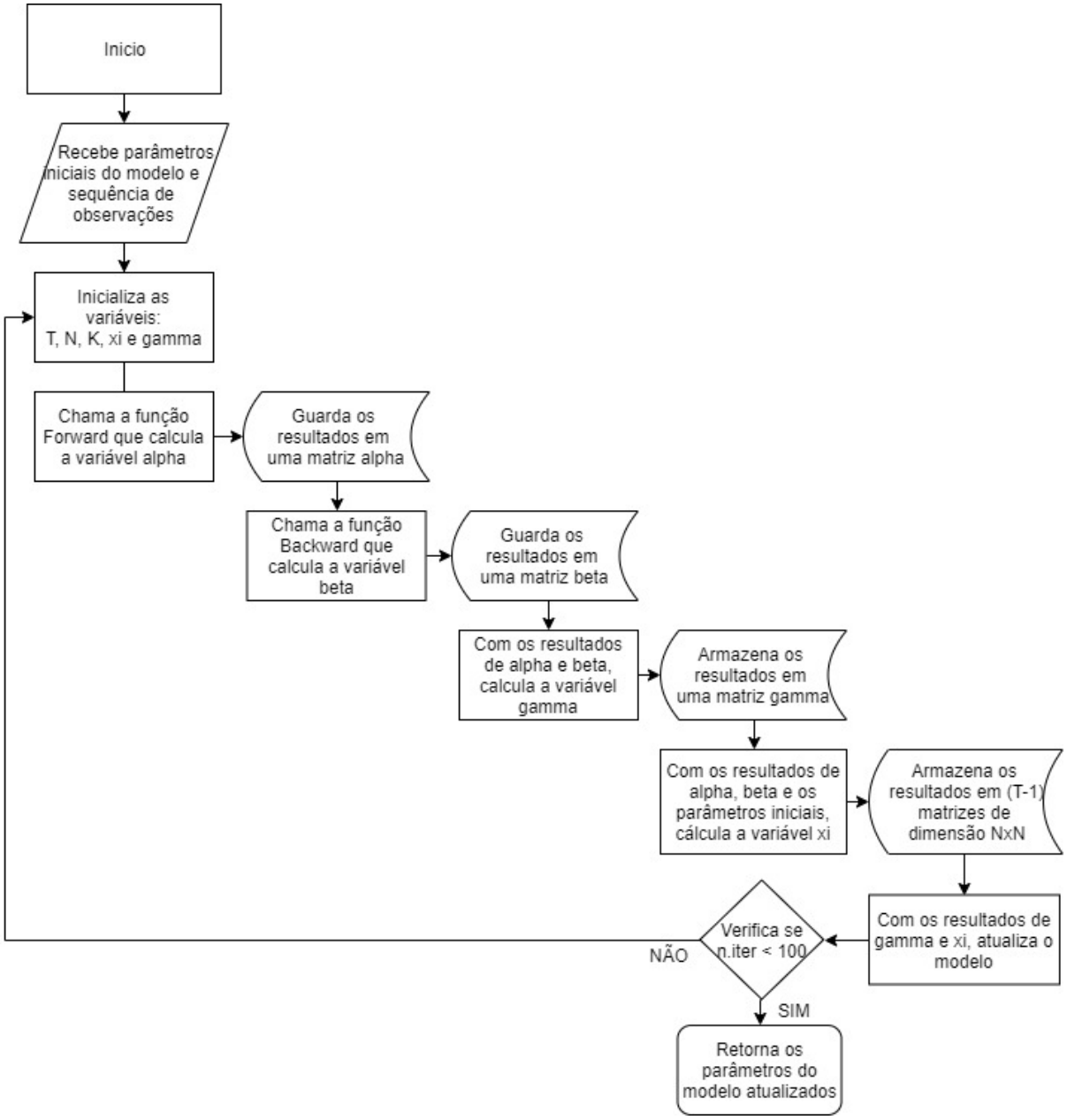
Por fim, atualiza-se o parâmetro *B*, entre as linhas 34 e 37. Esta atualização é realizada através da divisão da linha 37, onde o numerador é o número esperado de vezes em que o sistema está no estado *j* e se enxerga a observação v_k (onde *k* é o número de símbolos observáveis diferentes), e o denominador é o número esperado de vezes neste estado *j*. Então, o numerador é dado pela soma da coluna da matriz *gamma* que representa o estado em questão, onde se tem a observação v_k. E o denominador é dado pela soma de todos os elementos da coluna da matriz *gamma*, a qual representa o estado sendo avaliado.

Finalmente, depois de repetir os passos da função *Baum-Welch* pelo número de vezes estipulado pela variável *n.iter*, os valores dos parâmetros λ = (A, B e π) são retornados e tem-se a melhor resposta. Um fluxograma da função *Baum-Welch* é demonstrado na Figura 11.

Agora, será feito o detalhamento do funcionamento das funções *Forward* e *Backward*, as quais são chamadas dentro da função *Baum-Welch.*

A função *Forward* fornece a probabilidade de estar em um estado *i*, no tempo *t*, dada a sequência de observações que vieram antes do tempo *t* em questão. No pseudocódigo Algoritmo 5 é possível observar a implementação desta função. Na linha 1, a função recebe os parâmetros do modelo λ = (A, B e π), e também a sequência de observações *O*.

Figura 11 - Fluxograma da Função Baum-Welch.



Fonte: Elaborado pela autora.

Nas linhas 3 a 6, as variáveis são inicializadas. *Alpha* será uma matriz de dimensão (*T*x*N*), onde *T* representa cada passo de tempo *t* da sequência de observações, e *N* representa o número de estados escondidos diferentes do modelo. Também são inicializadas as variáveis *alphabeforeS* e *alphaafterS*, as quais servem para guardar os valores de antes e depois de realizar o processo de escalamento da variável *alpha*. É necessário realizar este processo a fim de evitar o problema de *underflow,* visto que se está trabalhando com uma quantidade grande de dados.

Algoritmo 5 – Função Forward

```
1:  FUNÇÃO forward(v, a, b, initial_distribution)
2:
3:    VAR
4:          T, m: numérico
5:          alpha, alphabeforeS, alphaafterS: matriz
6:          Scoef: vetor
7:
8:    PARA (i = 1 até m) FAÇA
9:        alpha[1, i] = initial_distribution[i]*b[i, v[1]]
10:       alphabeforeS[1, ] = alpha[1, ]
11:       Scoef[1] = 1/(sum(alpha[1, ]))
12:       alphaafterS[1, ] = Scoef[1]*alpha[1, ]
13:   FimPARA
14:
15:   PARA (t = 2 até T) FAÇA
16:       tmp = alphaafterS[t-1, ] %*% a
17:       alphabeforeS[t, ] = tmp * b[, v[t]]
18:       Scoef[t] = 1/(sum(alphabeforeS[t, ]))
19:       alphaafterS[t, ] = Scoef[t] * alphabeforeS[t, ]
20:   FimPARA
21:
22:   Retorna a variável alphaafterS
23:  FimFUNÇÃO
```

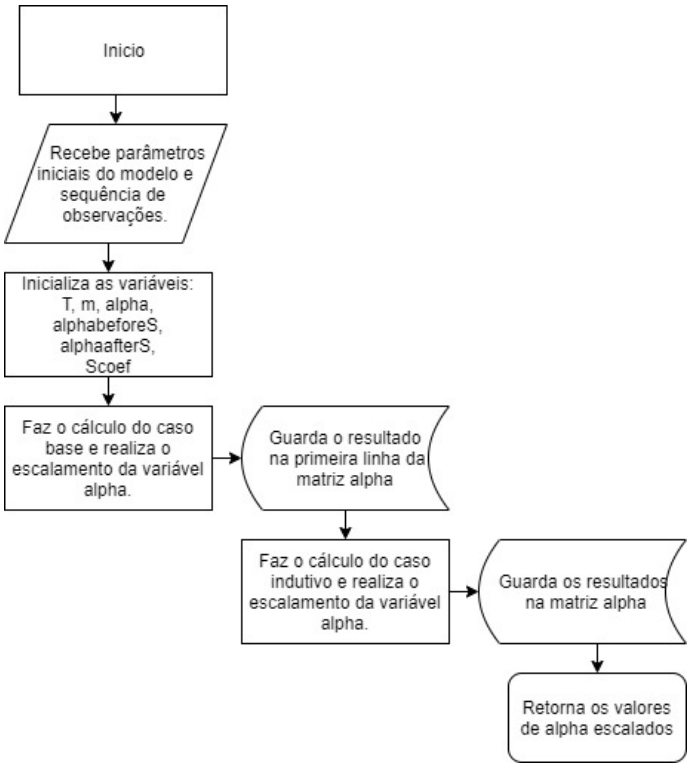
Entre as linhas 8 e 13 é realizada a etapa de inicialização da função *forward,* onde o cálculo do caso base é realizado, e também é aplicado o processo de escalamento da variável *alpha*, dado pelas Equações 22, 23, 24 e 25. Neste passo, toma-se a distribuição de probabilidade inicial do modelo (π), e multiplica-se pela probabilidade de se ter a observação O_1 em cada um dos estados escondidos existentes. Esta etapa fornece a probabilidade de estar em cada um desses estados, no tempo t=1, dada a observação vista neste tempo.

Em seguida, nas linhas 15 a 20, é realizado o cálculo do passo indutivo juntamente com o processo de escalamento da variável *alpha*, dado pelas Equações 26, 27 e 28. Esta etapa fornece o restante dos valores de *alpha*. O cálculo do *forward* para qualquer estado que não seja o inicial deve levar em consideração todas as formas possíveis de ter chegado a este estado.

Assim, o que a função *Forward* faz é utilizar a probabilidade computada de estar em cada um dos estados, em um passo de tempo atual, e com isso calcular a probabilidade do próximo passo de tempo *t*.

Por fim, na linha 22 a variável *alpha* é retornada após o escalamento, assim cada linha da matriz *alpha* representa um passo de tempo *t*, e cada coluna representa um dos estados existentes. Ou seja, cada elemento desta matriz diz qual a probabilidade de se estar em um determinado estado, em um tempo *t*, dadas as observações que vieram antes desse tempo. Na Figura 12 é possível visualizar o fluxograma da função *Forward.*

Figura 12 - Fluxograma da Função Forward.



Fonte: Elaborado pela autora.

A função *Backward* calcula a probabilidade de se ter uma dada sequência de observações no futuro, a partir de $t + 1$, sabendo que no tempo *t*, o sistema se encontra no estado *i*. No cálculo da variável β percorre-se o mesmo caminho utilizado para o cálculo da variável α, porém, de trás para frente.

No pseudocódigo Algoritmo 6 é possível observar a implementação desta função. Na linha 1, a função recebe os parâmetros do modelo λ = (*A* e *B*), e também a sequência de observações *O*. Não há necessidade de passar o parâmetro π, visto que se realiza o cálculo a partir da ultima observação da sequência.

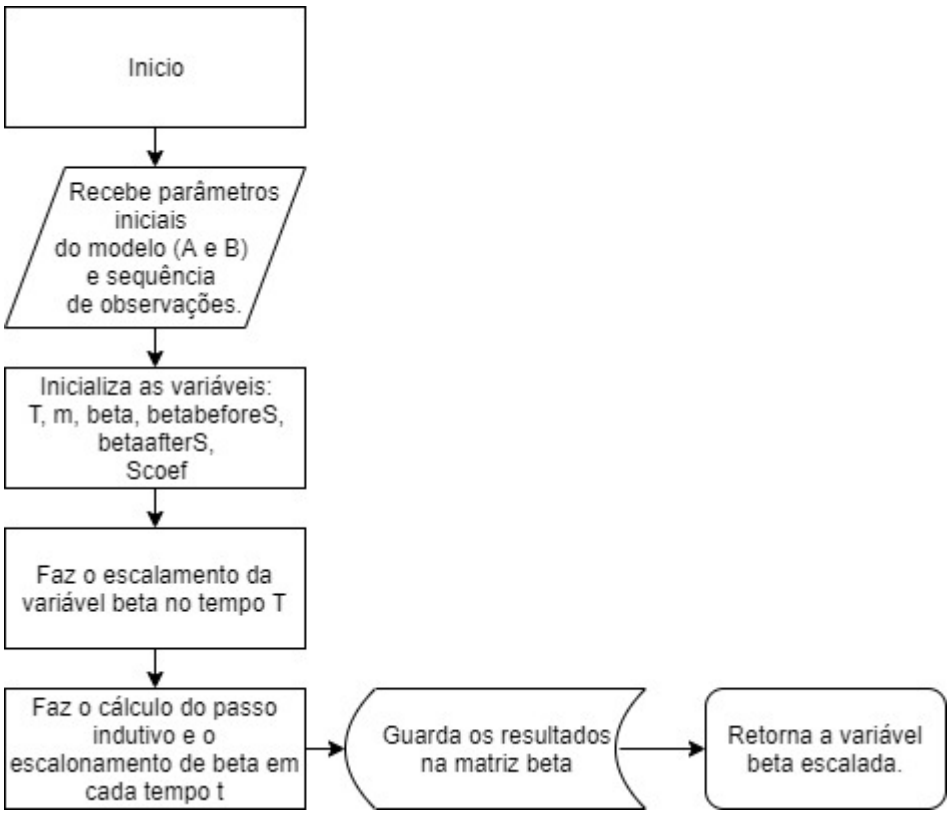
Em seguida, nas linhas 3 a 6, as variáveis *T*, *m* e *beta* são inicializadas. Também são inicializadas as variáveis *betabeforeS*, *betaafterS* e *Scoef*, as quais são utilizadas para realizar o escalamento da variável *beta*. Não é necessário calcular o caso base, pois nesse caso, ele terá sempre probabilidade igual a 1, já que quando o sistema estiver no último passo de tempo *T*, não existirão observações futuras. Então, na linha 8, o escalamento da variável *beta* é realizado diretamente através da multiplicação pelo coeficiente de escalamento.

Algoritmo 6 – Função Backward

```
1:  FUNÇÃO Backward(v, a, b)
2:
3:    VAR
4:          T, m: numérico
5:          beta, betabeforeS, betaafterS: matriz
6:          Scoef: vetor
7:
8:    betaafterS[T, ] = Scoef[T]*beta[T, ]
9:
10:   PARA (t = T-1 até 1) FAÇA
11:       tmp = as.matrix(betaafterS[t+1, ] * b[, v[t+1]])
12:       betabeforeS[t, ] = t(a %*% tmp)
13:       betaafterS[t, ] = Scoef[t]*betabeforeS[t, ]
14:   FimPARA
15:
16:   Retorna a variável betaafterS
17:  FimFUNÇÃO
```

E o passo indutivo, juntamente ao processo de escalamento desta variável, dado pelas Equações 32 e 33, é calculado entre as linhas 10 e 14. Ao final, a função retorna a variável *beta* depois de escalada, a qual também será uma matriz de dimensão (*T*x*N*), onde *T* é o número de observações visualizadas, e representa cada passo de tempo *t*, e *N* é o número de estados escondidos do modelo. Então, cada elemento desta matriz diz qual a probabilidade do sistema estar em um determinado estado, em um tempo *t*, sabendo quais as observações que ainda serão vistas no futuro. Na Figura 13 é possível visualizar o fluxograma da função *Backward.*

Figura 13 - Fluxograma da Função Backward.



Fonte: Elaborado pela autora.

3.3 Método de Extração de Parâmetros Baseado na Discretização de Medidas (Método B)

Este método tem como objetivo isolar o RTN do seu sinal de origem, e também caracteriza-lo para interpretações e análises futuras. A metodologia utilizada para a realização da seleção dos ruídos envolveu a elaboração de um algoritmo responsável por identificar picos de corrente da ordem de grandeza desse sinal, e, após isso, mensurar o tempo em captura e emissão desses ruídos. Após essa etapa, tem-se o RTN caracterizado e bem definido.

Em um primeiro momento, o algoritmo faz a discretização das medidas, isto é, torna as transições das armadilhas discretas. Assim, a variação de corrente que aconteceria em mais de um ponto ocorre de um ponto ao outro, de forma que seja possível determinar onde a transição de estado da armadilha começa e termina. Logo em seguida, as variáveis geradas podem ser usadas para extrair os tempos do RTN, ou seja, a segunda etapa irá extrair os tempos de captura e emissão da armadilha.

No pseudocódigo Algoritmo 7, é possível verificar a etapa de inicialização do algoritmo. Nesta etapa, o arquivo "Data.csv" é lido, e seus valores são guardados em uma matriz denominada *data*. A primeira coluna desta matriz guarda os valores de tempo, e na segunda coluna são armazenados os valores do sinal RTS gerado. A variável *Id* recebe os valores do sinal RTS gerado e a variável *Tempo* recebe os valores de tempo, o qual possui um $\Delta t = 1s$, como já foi mencionado anteriormente.

A variável *aRuido* é definida de forma a determinar o nível máximo ao qual os pontos de transição serão juntados. Outras variáveis que também serão utilizadas no desenvolvimento do código são inicializadas nesse momento.

Após a inicialização das variáveis necessárias, a variável *DeltaId* armazena os valores que são dados pela diferença do valor do sinal RTS no tempo $t + 1$ e o valor desse sinal no tempo t. Esse processo está descrito entre as linhas 7 e 11 do pseudocódigo Algoritmo 7.

Algoritmo 7 – Inicialização

```
1:  data ← arquivo de dados Data.csv
2:
3:  VAR
4:      aRuido, nmedidas, IdMax, zn, yn: numérico
5:      Id, Tempo, DeltaId, IdBuild, DeltaIdBuild, IdRTN, Deltat, tcap, temi: matriz
6:
7:  PARA (n = 1 até nmedidas) FAÇA
8:      PARA (m = 1 até length(Id)-1) FAÇA
9:          DeltaId[m,n] = Id[m+1,n] - Id[m,n]
10:     FimPARA
11:  FimPARA
```

Como consequência do código de discretização, picos ou vales de correntes constituídos de apenas um ponto acabam por desaparecer da solução final. Tendo isso em vista, é preciso criar uma etapa dentro deste algoritmo que detecte esses picos e vales e não execute o código de discretização nesses pontos. Como exemplificado entre as linhas 12 e 53 do pseudocódigo Algoritmo 8, esse processo foi feito através da criação das variáveis *testepiconegativo* e *testepicopositivo*, as quais recebem os valores 0 ou 1, e servem para verificar se há um pico negativo ou um pico positivo no sinal que está sendo analisado.

Neste passo é preciso verificar o valor armazenado na variável *DeltaId* e realizar uma comparação com o valor da variável *aRuido*. Para o primeiro valor de *DeltaId*, em t=1, tem-se que a variável *testepiconegativo* recebe o valor 0, caso o valor armazenado em *DeltaId* seja menor do que o valor de *aRuido*. E a variável *testepicopositivo* recebe o valor 0, caso este valor de *DeltaId* seja maior do que *–aRuido*.

Para os demais valores de *DeltaId*, a partir do tempo t=2, é preciso olhar para seu valor no tempo t, e para seu valor no tempo t-1. Neste caso, se o valor de *DeltaId*, no tempo t, for maior do que o valor de *aRuido*, e o valor de *DeltaId*, no tempo t-1, for maior do que o valor de *–aRuido*, então a variável *testepiconegativo* recebe o valor 0. E se o valor de *DeltaId*, no tempo t, for maior do que o valor de *aRuido*, e o valor de *DeltaId*, no tempo t-1, for menor do que o valor de *–aRuido*, então a variável *testepiconegativo* recebe o valor 1.

Seguindo o mesmo raciocínio, pode-se atribuir os valores para a variável *testepicopositivo*. Neste caso, se o valor de *DeltaId*, no tempo *t*, for menor do que o valor de *–aRuido*, e o valor de *DeltaId*, no tempo *t*-1, for menor do que o valor de *aRuido*, então a variável *testepicopositivo* recebe o valor 0. E se o valor de *DeltaId*, no tempo *t*, for menor do que o valor de *–aRuido*, e o valor de *DeltaId*, no tempo *t*-1, for maior do que o valor de *aRuido*, então a variável *testepicopositivo* recebe o valor 1.

Posteriormente, entre as linhas 55 e 102 do pseudocódigo é construída a variável *IdBuild*, a qual, ao final de todo processo, irá armazenar os valores do sinal RTS. Para isso, tem-se a variável *IdMax* que em um primeiro momento armazenará os valores deste sinal RTS gerado. Em seguida, é feita uma comparação com os valores armazenados nas variáveis *testepiconegativo* e *testepicopositivo*. Se o valor de *testepicopositivo* for 0, a variável *IdBuild* irá receber o valor mínimo armazenado em *IdMax*. E caso o valor de *testepiconegativo* for 0, então a variável *IdBuild* irá receber o valor máximo armazenado em *IdMax*.

Algoritmo 8 – Discretização de Medidas

```
1:  VAR
2:        aux, cont, passoucontador ← 0
3:
4:  PARA (n = 1 até nmedidas) FAÇA
5:        m=1
6:        WHILE (m < tamanho de data) FAÇA
7:
8:              aux, aux1 ← 0
9:              minicial ←1
10:             cont ← 1
11:
12:             SE (aux<1)
13:               SE (DeltaId[m,n] > aRuido)
14:                 SE(m>1 && m<(length(data$RTS)-1))
15:                   SE(DeltaId[m-1,n] > -aRuido)
16:                     testepiconegativo=0
17:                   FimSE
18:                   SENAO
19:                     testepiconegativo=1
20:                   FimSENAO
21:                 FimSE
22:                 SENAO
23:                     testepiconegativo=0
24:                 FimSENAO
25:                 WHILE (DeltaId[m,n] > aRuido && m < (length(data$RTS)-1)) FAÇA
```

```
26:                         cont = m + 1
27:                         m = m+1
28:                         aux = 1
29:                     FimWHILE
30:                 FimSE
31:             FimSE
32:
33:             SE (aux<1)
34:                 SE (DeltaId[m,n] < -aRuido)
35:                   SE(m>1 && m<(length(data$RTS)-1))
36:                     SE(DeltaId[m-1,n] < aRuido)
37:                         testepicopositivo=0
38:                     FimSE
39:                     SENAO
40:                         testepicopositivo=1
41:                     FimSENAO
42:                   FimSE
43:                   SENAO
44:                         testepicopositivo=0
45:                   FimSENAO
46                    WHILE (DeltaId[m,n] < 0 && m < (length(data$RTS)-1)) FAÇA
47:                         cont = m + 1
48:                         m = m+1
49:                         aux = 2
50:                         aux1 = 1
51:                   FimWHILE
52:                 FimSE
53:             FimSE
54:
55:             SE (aux > 0 && cont > minicial+1)
56:                 PARA(num=0 até (cont-minicial)) FAÇA
57:                       IdMax[num+1] = Id[minicial+num,n]
58:                 FimPARA
59:
60:                 SE (aux1>0)
61:                   SE (testepicopositivo==0)
62:                     PARA(a=minicial até cont) FAÇA
63:                           IdBuild[a,n] = min(IdMax)
64:                     FimPARA
65:                   FimSE
66:                   SENAO
67:                     IdBuild[minicial,n] = Id[minicial,n]
68:                     PARA(a=(minicial+1) até cont) FAÇA
69:                           IdBuild[a,n] = min(IdMax)
70:                     FimPARA
71:                   FimSENAO
72:                FimSE
73:                SENAO
74:                  SE (testepiconegativo==0)
75:                     PARA(a=minicial até cont) FAÇA
76:                           IdBuild[a,n] = max(IdMax)
77:                     FimPARA
78:                   FimSE
79:                   SENAO
80:                     IdBuild[minicial,n] = Id[minicial,n]
```

```
81:                         PARA(a=(minicial+1) até cont) FAÇA
82:                             IdBuild[a,n] = max(IdMax)
83:                         FimPARA
84:                       FimSENAO
85:                   FimSENAO
86:               rm(IdMax)
87:               IdMax ← 0
88:               FimSE
89:               SENAO
90:                  passoucontador = 0
91:                  SE (cont>1)
92:                     IdBuild[minicial,n] = Id[minicial,n]
93:                     m=minicial+1
94:                  FimSE
95:                  SENAO
96:                     IdBuild[m,n] = Id[m,n]
97:                     m=m+1
98:                  FimSENAO
99:               FimSENAO
100:       FimWHILE
101:       IdBuild[m,n] = Id[m,n]
102: FimPARA
```

Finalmente, tem-se a fase final da etapa de discretização de medidas, apresentada no pseudocódigo Algoritmo 9. Nesta etapa, tem-se a construção da variável *DeltaIdBuild*, a qual armazena os valores que são dados pela diferença do valor do sinal armazenado na variável *DeltaIdBuild* no tempo $t + 1$ e o valor desse sinal no tempo t. Entre as linhas 10 e 15 do pseudocódigo Algoritmo 9 é realizada a verificação se o valor armazenado em cada posição da variável *DeltaIdBuild* é maior do que 4e-4, e todos os valores que satisfazem esta condição são somados. Após realizar esta soma, divide-se este valor pelo número de vezes em que esta condição foi satisfeita, tendo assim o valor da média do histograma do sinal gerado.

Algoritmo 9 – Finalização da Discretização de Medidas

```
1:  PARA (n = 1 até nmedidas) FAÇA
2:       PARA(m = 1 até (length(data$RTS)-1)) FAÇA
3:            DeltaIdBuild[m,n] = IdBuild[m+1,n] - IdBuild[m,n]
4:       FimPARA
5:  FimPARA
6:
7:  Soma = 0
8:  Contador = 0
9:
10: PARA (i = 1 até (length(data$RTS)-1)) FAÇA
11:       SE (DeltaIdBuild[i] > 4e-4)
```

```
12:                 Soma=DeltaIdBuild[i]+Soma
13:                 Contador=Contador+1
14:           FimSE
15:  FimPARA
16:
17:  MediaHistograma=Soma/Contador
```

Logo após a etapa de discretização, inicia-se o processo para a extração dos parâmetros deste sinal, o qual pode ser visualizado no pseudocódigo Algoritmo 10. Em um primeiro momento são inicializadas as variáveis *LimInf* e *LimSup*, as quais recebem os valores que definem os limites da Gaussiana de *DeltaId* da armadilha. Em seguida, a variável *h* recebe todos os valores de *DeltaIdBuild*, os quais estão entre os valores de *LimInf* e *LimSup*, e fazendo a média dos valores de *h* tem-se o valor final da amplitude do sinal RTS gerado.

Entre as linhas 11 e 62 do pseudocódigo Algoritmo 10 é onde é montado o modelo do RTN, através da variável *IdRTN*, e também onde são armazenados os valores de *Tempo* em variáveis nomeadas como *tcap* e *temi*. A variável *MediaTrap*, a qual armazena o valor de *h*, é utilizada nesta etapa.

Se o valor armazenado na variável *DeltaIdBuild* for maior do que o valor de *LimInf* e menor do que o valor estipulado para *LimSup*, então a matriz *IdRTN*, em um tempo $t + 1$, recebe o valor de *IdRTN* em um tempo *t* somado ao valor da variável *MediaTrap*. Neste caso, a variável *tcap* irá receber o valor armazenado na variável *Tempo*, no momento *t*.

Se o valor armazenado na variável *DeltaIdBuild* for maior do que o valor de *–LimSup* e menor do que o valor estipulado para *–LimInf*, então a matriz *IdRTN*, em um tempo $t + 1$, recebe a diferença entre o valor de *IdRTN* em um tempo *t* e o valor da variável *MediaTrap*. Neste caso, a variável *temi* irá receber o valor armazenado na variável *Tempo*, no momento *t*.

Em seguida, entre as linhas 64 e 78, são criadas as variáveis *tcaptura* e *temissao*, as quais recebem os valores de *tcap* e *temi*, respectivamente. Ao final, o valor do tempo de

captura da armadilha é dado pela média da variável *tcaptura*, e o valor do tempo de emissão desta armadilha é dado pela média da variável *temissao*.

Algoritmo 10 – Extração de Parâmetros

```
1:
2:   VAR
3:        LimInf, LimSup: numérico
4:
5:   logicalIndexes ← DeltaIdBuild
6:   h ← logicalIndexes[logicalIndexes < LimSup & DeltaIdBuild > LimInf]
7:
8:   meanValue = mean(h)
9:   MediaTrap = meanValue
10:
11:  PARA (n = 1 até nmedidas) FAÇA
12:       terminou, z, y, passouemi, passoucap, indicecap, indiceemi, aux ← 0
13:       IdRTN[1,n] = IdBuild[1,n]
14:
15:       PARA (m=1 até (length(data$RTS)-1)) FAÇA
16:            aux=0
17:            SE (DeltaIdBuild[m,n] > LimInf && DeltaIdBuild[m,n] < LimSup)
18:               passouemi = 1
19:               SE (y==0 || passoucap>0)
20:                    z = z+1
21:                    tcap[z,n]=Tempo[m,n]
22:                    IdRTN[m+1,n] = IdRTN[m,n] + MediaTrap
23:                    aux=1
24:               FimSE
25:
26:               SE (passoucap>0)
27:                    temi[y,n]=Tempo[m,n]-temi[y,n]
28:                    terminou = 1
29:               FimSE
30:               passoucap = 0
31:            FimSE
32:
33:            SE (DeltaIdBuild[m,n] > -LimSup && DeltaIdBuild[m,n] < -LimInf)
34:               passoucap = 1
35:               SE (z==0 || passouemi>0)
36:                    y = y+1
37:                    temi[y,n]=Tempo[m,n]
38:                    IdRTN[m+1,n] = IdRTN[m,n] - MediaTrap
39:                    aux=1
40:               FimSE
41:
42:               SE (passouemi>0)
43:                    tcap[z,n]=Tempo[m,n]-tcap[z,n]
44:                    terminou = 0
45:               FimSE
46:               passouemi = 0
47:            FimSE
48:
49:            SE (aux<1)
```

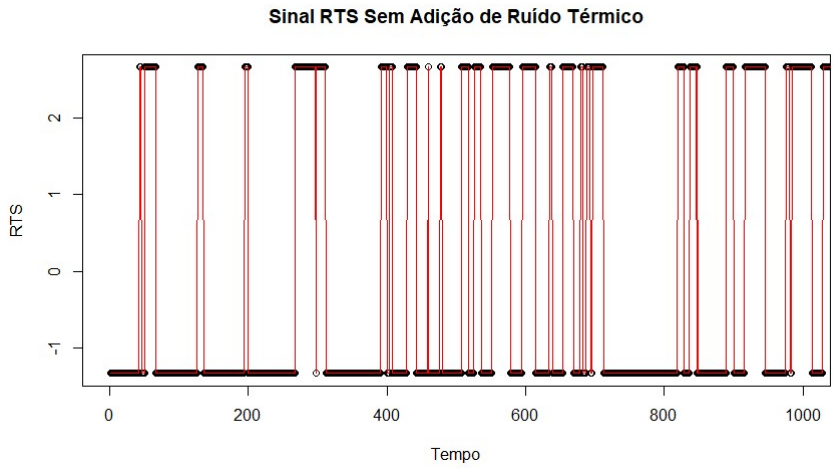
```
50:                     IdRTN[m+1,n] = IdRTN[m,n]
51:                 FimSE
52:         FimPARA
53:
54:         SE (terminou>0)
55:            zn[n] = z-1
56:            yn[n] = y
57:         FimSE
58:         SENAO
59:            zn[n] = z
60:            yn[n] = y-1
61:         FimSENAO
62:   FimPARA
63:
64:   VAR
65:       tcaptura, temissao: vetor
66:       temp, tempo ← 1
67:
68:    PARA (n = 1 até nmedidas) FAÇA
69:          PARA (m = 1 até zn[n]) FAÇA
70:                tcaptura[temp] = tcap[m,n]
71:                temp = temp + 1
72:          FimPARA
73:
74:          PARA (m = 1 até yn[n]) FAÇA
75:                temissao[tempo]=temi[m,n]
76:                tempo = tempo + 1
77:          FimPARA
78:     FimPARA
79:
80:    PARA (n = 1 até nmedidas) FAÇA
81:          PARA (m in 1:(length(data$RTS)-1)) FAÇA
82:                Deltat[m,n] = Tempo[m+1,n] - Tempo[m,n]
83:          FimPARA
84:     FimPARA
85:
86:     MaiorMatriz= max(Deltat)
87:     MaiorDeltaT = max(MaiorMatriz)
88:
89:     toff = mean(temissao)
90:     ton = mean(tcaptura)
```

4 RESULTADOS

Neste capítulo os resultados obtidos com os algoritmos desenvolvidos são apresentados e discutidos. Em um primeiro momento são mostrados os resultados do algoritmo de geração do RTN sintético, tanto com o sinal limpo, quanto com a adição de ruído térmico ao sinal. Em seguida são apresentados os resultados gerados pelos dois métodos de extração de parâmetros utilizados neste trabalho. Por fim, é realizada a análise dos resultados, fazendo uma comparação de desempenho entre os métodos de extração de parâmetros desenvolvidos. A comparação dos resultados é feita através da análise dos tempos de captura e emissão extraídos por cada um dos métodos, e também pela extração da amplitude ΔI. Estes resultados estão apresentados nas Tabelas de 1 a 4.

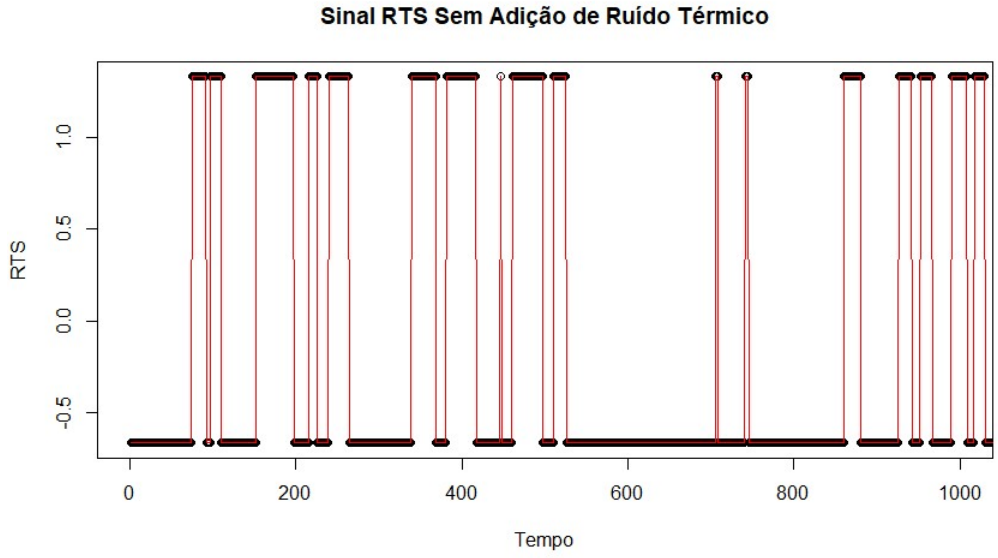
Inicialmente é gerado um sinal RTS sintético limpo, isto é, sem a adição do ruído térmico. Para estes testes foram gerados gráficos para um sinal RTS de amplitude igual a 4, tempo de captura (τ_c) igual a 10 segundos e tempo de emissão (τ_e) igual a 20 segundos, e também para um sinal RTS de amplitude igual a 2, tempo de captura (τ_c) igual a 20 segundos e tempo de emissão (τ_e) igual a 40 segundos. Os sinais gerados por estes parâmetros podem ser visualizados nas Figuras 14 e 15, respectivamente.

Figura 14 - Sinal RTS de Ampl = 4, $\tau_c = 10s$, e $\tau_e = 20s$.



Fonte: Elaborado pela autora.

Figura 15 - Sinal RTS de Ampl = 2, $\tau_c = 20s$, e $\tau_e = 40s$.



Fonte: Elaborado pela autora.

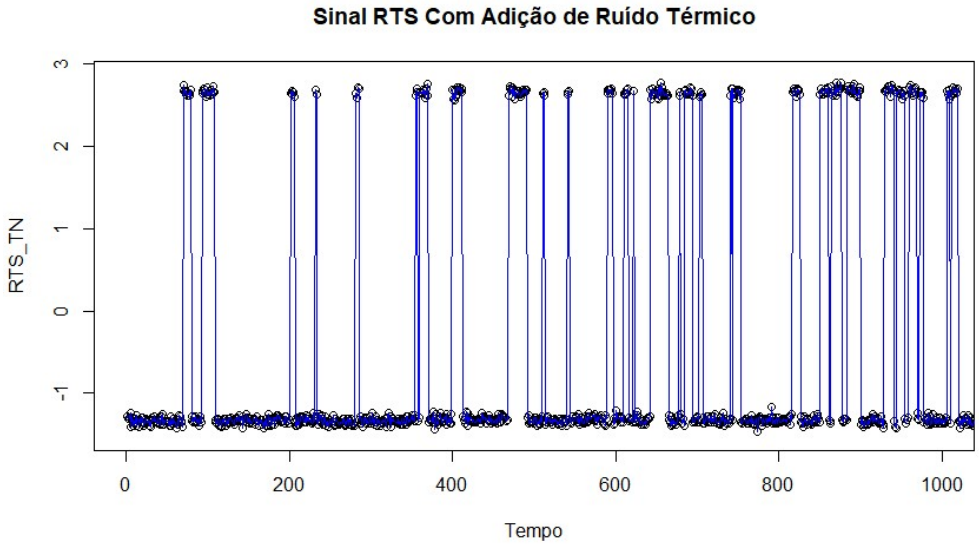
Sabendo que ambos os sinais são gerados por apenas uma armadilha, pode-se verificar que ambos os gráficos apresentam dois níveis bem definidos e que a diferença entre estes níveis fornece o valor da amplitude do sinal gerado. Também se pode notar que existem transições ocorrendo entre os valores observáveis distintos gerados. Pode-se concluir que estes sinais são gerados com base nos parâmetros determinados para cada um deles.

Em um segundo momento, é gerado um sinal RTS com adição de ruído térmico. Para estes testes foram gerados gráficos para um sinal RTS de amplitude igual a 4, tempo de captura (τ_c) igual a 10 segundos e tempo de emissão (τ_e) igual a 20 segundos. Neste sinal foi adicionado um ruído térmico com desvios-padrão (sd) equivalentes a 1%, 10% e 50% da amplitude do sinal RTS gerado. Os sinais gerados por estes parâmetros podem ser visualizados nas Figuras 16, 17 e 18, respectivamente.

Na Figura 16 é possível verificar que mesmo adicionando um ruído térmico com desvio-padrão igual a 0.04, isto é, equivalente a 1% da amplitude do sinal RTS, ainda se tem níveis bem definidos diferenciando os valores observáveis distintos gerados por estes

parâmetros. Logo, pode-se notar que o sinal sofre pouca distorção com a adição de um ruído térmico desta magnitude.

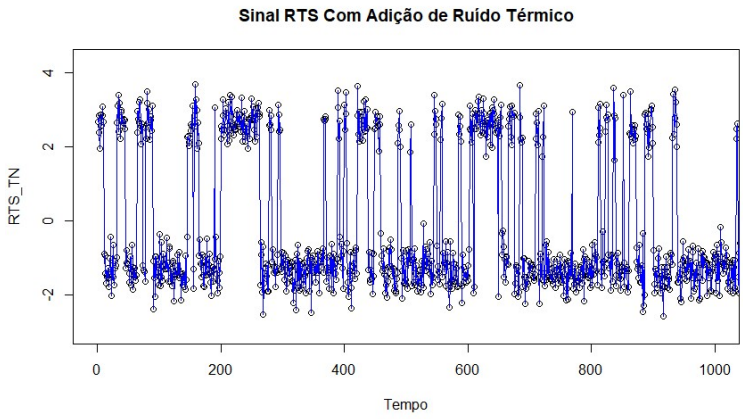
Figura 16 - Sinal RTS de Ampl = 4, $\tau_c = 10s$, e $\tau_e = 20s$ e $sd = 0.04$.



Fonte: Elaborado pela autora.

Já na Figura 17 é adicionado ao sinal um ruído térmico com desvio-padrão igual a 0.4, isto é, equivalente a 10% da amplitude do sinal RTS. Nota-se que embora ainda seja possível visualizar os níveis diferentes gerados pelos parâmetros estipulados, o sinal sofre uma distorção maior.

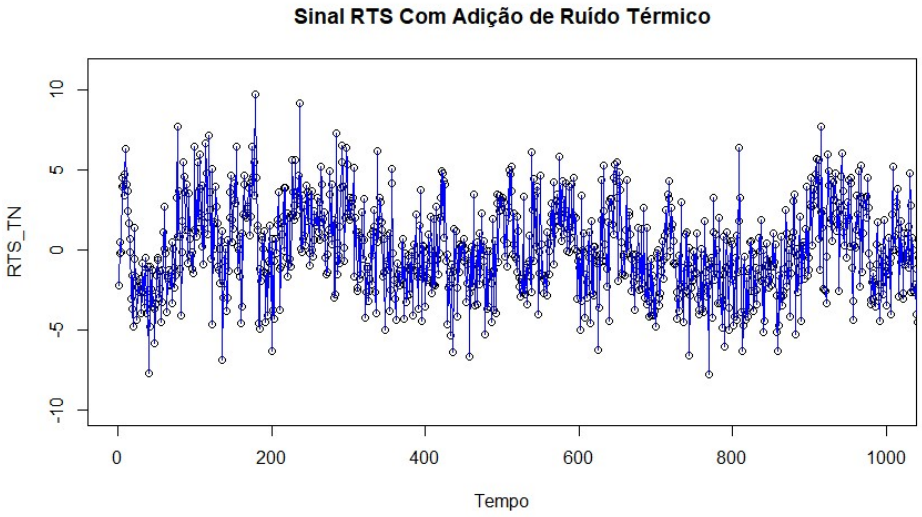
Figura 17 - Sinal RTS de Ampl = 4, $\tau_c = 10s$, e $\tau_e = 20s$ e $sd = 0.4$.



Fonte: Elaborado pela autora.

Por último, na Figura 18 é adicionado ao sinal um ruído térmico com desvio-padrão igual a 2, isto é, equivalente a 50% da amplitude do sinal RTS. Neste caso, pode-se notar a dificuldade em visualizar com clareza os níveis distintos gerados pelos parâmetros aplicados. O sinal sofre uma distorção muito maior, não sendo possível identificar visualmente qual a amplitude real do ruído RTN.

Figura 18 - Sinal RTS de Ampl = 4, $\tau_c = 10s$, e $\tau_e = 20s$ e $sd = 2$.



Fonte: Elaborado pela autora.

Após verificar o funcionamento dos algoritmos de geração do sinal RTS limpo e com a adição de ruído térmico, foram realizados testes com os métodos desenvolvidos de extração de parâmetros.

Em um primeiro momento foram feitos testes com o sinal RTS limpo. Na Tabela 1 é possível ver os valores dos parâmetros extraídos pelo método A, o qual é baseado no algoritmo de *Baum-Welch*. E na Tabela 2, encontram-se os resultados dos parâmetros extraídos pelo método B.

Com base nos resultados apresentados em ambas as Tabelas, conclui-se que os dois métodos conseguem extrair corretamente os parâmetros que geraram o sinal RTS em análise. Na Tabela 1, ainda é possível verificar que conforme os valores de tempo de captura (τ_c) e

tempo de emissão (τ_e) aumentam, a precisão dos valores extraídos destes parâmetros diminui. Isso ocorre devido ao fato de que este método se baseia nas transições que ocorrem no sinal RTS gerado, e conforme os valores de τ_c e τ_e aumentam o número de transições que ocorrem diminuem, logo os valores obtidos se tornam menos precisos. Porém, pode-se resolver este problema aumentando-se o número de amostras do sinal que será analisado.

Tabela 1 - Resultados da extração de parâmetros do sinal RTS limpo através do método A.

Parâmetros Gerados	**Parâmetros Extraídos**
Ampl = 4 Tc = 10 Te = 20	Ampl = 4 Tc = 10.0805 Te = 20.0441
Ampl = 4 Tc = 40 Te = 60	Ampl = 4 Tc = 40.8561 Te = 60.3256
Ampl = 4 Tc = 100 Te = 200	Ampl = 4 Tc = 97.7737 Te = 199.9617
Ampl = 4 Tc = 400 Te = 600	Ampl = 4 Tc = 362.9574 Te = 617.9328

Tabela 2 - Resultados da extração de parâmetros do sinal RTS limpo através do método B.

Parâmetros Gerados	**Parâmetros Extraídos**
Ampl = 4 Tc = 10 Te = 20	Ampl = 4 Tc = 9.9596 Te = 20.0193
Ampl = 4 Tc = 40 Te = 60	Ampl = 4 Tc = 39.8092 Te = 59.1673
Ampl = 4 Tc = 100 Te = 200	Ampl = 4 Tc = 99.4109 Te = 214.8207
Ampl = 4 Tc = 400 Te = 600	Ampl = 4 Tc = 398.1794 Te = 597.4883

Após foram realizados testes com o sinal RTS com a adição do ruído térmico. Na Tabela 3 é possível observar os valores dos parâmetros extraídos pelo método A. Nestes testes foi considerada a adição de ruído térmico com diversos valores de desvios-padrão diferentes.

Os valores estipulados para os parâmetros que geram o sinal RTS em análise foram uma amplitude de 4, um tempo de captura igual a 10 e um tempo de emissão igual a 20.

Tabela 3 - Resultados da extração de parâmetros do sinal RTS com adição de ruído térmico através do método A.

Parâmetros Gerados	Ampl = 4 Tc = 10; Te = 20
Parâmetros Extraídos (sd = 0.004)	Ampl = 3.9999 Tc = 9.9227; Te = 20.0233
Parâmetros Extraídos (sd = 0.006)	Ampl = 4.0000 Tc = 10.0513; Te = 20.3586
Parâmetros Extraídos (sd = 0.008)	Ampl = 3.9999 Tc = 9.9375; Te = 20.0989
Parâmetros Extraídos (sd = 0.01)	Ampl = 4.0000 Tc = 10.0263; Te = 19.6732
Parâmetros Extraídos (sd = 0.02)	Ampl = 3.9999 Tc = 9.9439; Te = 20.4449
Parâmetros Extraídos (sd = 0.04)	Ampl = 3.9999 Tc = 10.1249; Te = 20.0955
Parâmetros Extraídos (sd = 0.06)	Ampl = 4.0002 Tc = 10.0773; Te = 20.0838
Parâmetros Extraídos (sd = 0.08)	Ampl = 3.9999 Tc = 9.9802; Te = 20.2257
Parâmetros Extraídos (sd = 0.1)	Ampl = 4.0003 Tc = 10.0455; Te = 20.2655
Parâmetros Extraídos (sd = 0.2)	Ampl = 4.0003 Tc = 10.0717; Te = 19.8665
Parâmetros Extraídos (sd = 0.4)	Ampl = 3.9976 Tc = 10.1171; Te = 19.9475
Parâmetros Extraídos (sd = 0.6)	Ampl = 3.9583 Tc = 9.8495; Te = 19.8134
Parâmetros Extraídos (sd = 0.8)	Ampl = 3.8823 Tc = 10.1013; Te = 20.3085
Parâmetros Extraídos (sd = 1)	Ampl = 3.8382 Tc = 9.9578; Te = 20.3923
Parâmetros Extraídos (sd = 1.2)	Ampl = 3.8850 Tc = 10.0143; Te = 19.8658
Parâmetros Extraídos (sd = 1.4)	Ampl = 3.9703 Tc = 9.7914; Te = 19.9575
Parâmetros Extraídos (sd = 1.6)	Ampl = 4.1154 Tc = 9.9741; Te = 19.9558
Parâmetros Extraídos (sd = 1.8)	Ampl = 4.3012 Tc = 9.9174; Te = 19.6439
Parâmetros Extraídos (sd = 2)	Ampl = 4.4834 Tc = 9.8645; Te = 20.2140
Parâmetros Extraídos (sd = 2.2)	Ampl = 4.6973 Tc = 9.9960; Te = 20.2523
Parâmetros Extraídos (sd = 2.4)	Ampl = 4.9277 Tc = 9.9228; Te = 19.6987
Parâmetros Extraídos (sd = 2.6)	Ampl = 5.1793 Tc = 9.9272; Te = 19.6823
Parâmetros	Ampl = 5.4259

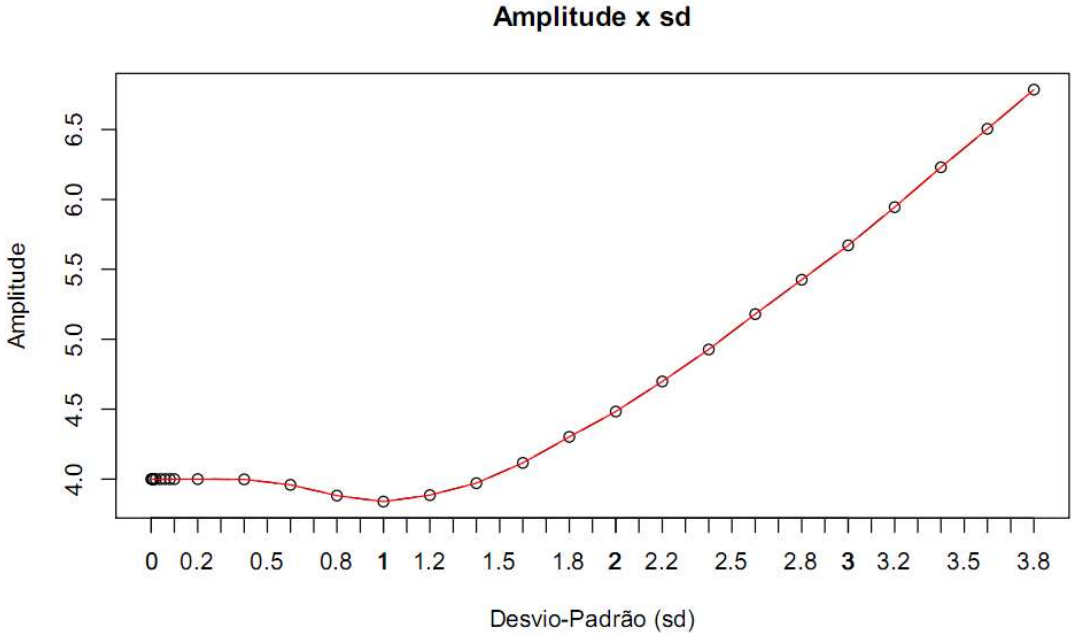
Extraídos (sd = 2.8)	Tc = 9.9436; Te = 20.2883
Parâmetros Extraídos (sd = 3)	Ampl = 5.6723 Tc = 9.6193; Te = 19.7569
Parâmetros Extraídos (sd = 3.2)	Ampl = 5.9452 Tc = 9.1318; Te = 16.8629
Parâmetros Extraídos (sd = 3.4)	Ampl = 6.2302 Tc = 7.6924; Te = 12.8711
Parâmetros Extraídos (sd = 3.6)	Ampl = 6.5044 Tc = 5.9855; Te = 8.0906
Parâmetros Extraídos (sd = 3.8)	Ampl = 6.7843 Tc = 4.3566; Te = 4.6614

Através dos resultados apresentados na Tabela 3 conclui-se que para um sinal RTS com adição de um ruído térmico, cujo desvio-padrão é equivalente até 40% da amplitude do sinal RTS gerado, o método elaborado neste trabalho faz a extração dos parâmetros deste sinal corretamente. Quando o desvio-padrão do ruído térmico atinge o valor de 50% da amplitude do sinal RTS, o valor do parâmetro que fornece a amplitude deste sinal se torna menos preciso, porém os valores de tempo de captura e emissão que são extraídos pelo algoritmo continuam próximos ao valor real, ou seja, o método ainda é capaz de fornecer os valores de τ_c e τ_e do sinal RTS gerado.

Ainda, com base na Tabela 3 é possível destacar que o método A se torna incapaz de extrair os valores dos parâmetros do sinal RTS quando o desvio-padrão do ruído térmico adicionado a este sinal atinge um valor de 3.2 (isto é, 80% da amplitude do sinal RTS gerado).

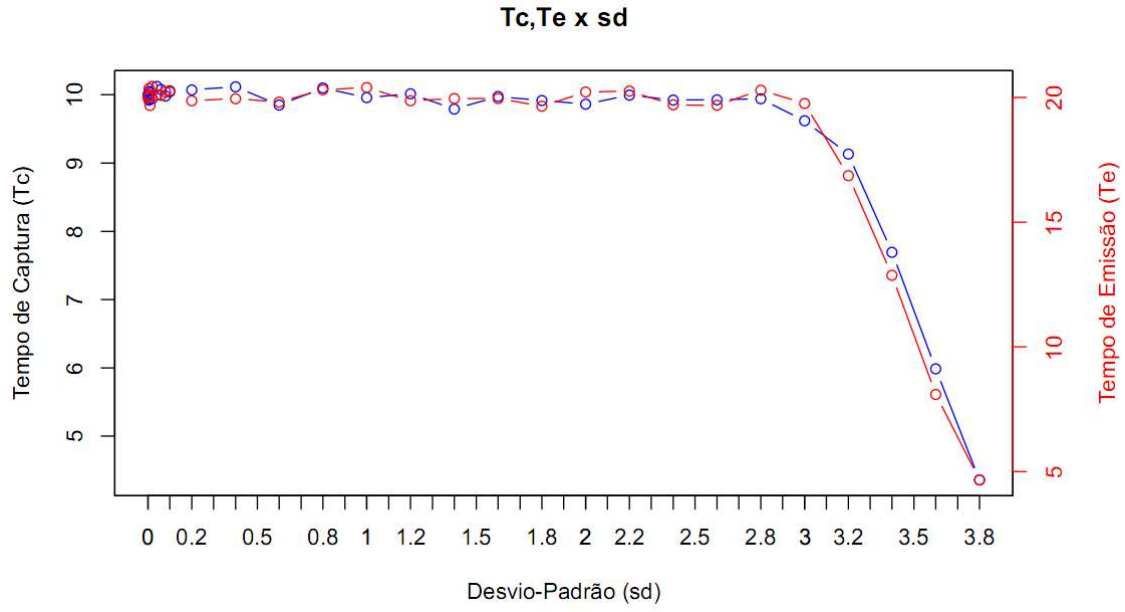
Para uma melhor visualização dos resultados da Tabela 3, estes foram representados através dos gráficos mostrados nas Figuras 19 e 20. O gráfico da Figura 19 apresenta uma comparação entre a amplitude do sinal e o desvio-padrão aplicado ao ruído térmico adicionado a este sinal. Enquanto o gráfico da Figura 20 mostra a comparação entre os parâmetros de tempo de captura e tempo de emissão deste sinal com o valor do desvio-padrão utilizado. Com base nos gráficos é possível confirmar como os parâmetros extraídos pelo método se distanciam dos valores gerados, conforme o desvio-padrão do ruído aumenta.

Figura 19 - Resultados da Tabela 3 de Amplitude x sd.



Fonte: Elaborado pela autora.

Figura 20 - Resultados da Tabela 3 de Tc, Te x sd.



Fonte: Elaborado pela autora.

Finalmente, na Tabela 4 é possível observar os valores dos parâmetros extraídos pelo método B. Com o objetivo de realizar comparações de desempenho entre os dois métodos, foram utilizados os mesmos valores usados nos testes feitos com o método anterior.

Tabela 4 - Resultados da extração de parâmetros do sinal RTS com adição de ruído térmico através do método B.

Parâmetros Gerados	Ampl = 4 Tc = 10; Te = 20
Parâmetros Extraídos (sd = 0.004)	Ampl = 4.0055 Tc = 10.0644; Te = 19.6656
Parâmetros Extraídos (sd = 0.006)	Ampl = 4.0080 Tc = 10.0092; Te = 19.6258
Parâmetros Extraídos (sd = 0.008)	Ampl = 4.0107 Tc = 10.0916; Te = 19.7434
Parâmetros Extraídos (sd = 0.01)	Ampl = 4.0134 Tc = 10.0513; Te = 20.4600
Parâmetros Extraídos (sd = 0.02)	Ampl = 4.0270 Tc = 10.0483; Te = 19.7497
Parâmetros Extraídos (sd = 0.04)	Ampl = 4.0538 Tc = 10.0344; Te = 19.6630
Parâmetros Extraídos (sd = 0.06)	Ampl = 4.0804 Tc = 10.0623; Te = 20.0009
Parâmetros Extraídos (sd = 0.08)	Ampl = 4.1061 Tc = 10.0156; Te = 19.8670
Parâmetros Extraídos (sd = 0.1)	Ampl = 4.0899 Tc = 10.1373; Te = 19.7939
Parâmetros Extraídos (sd = 0.2)	Ampl = 4.2288 Tc = 10.3080; Te = 19.9858
Parâmetros Extraídos (sd = 0.4)	Ampl = 4.3751 Tc = 10.1639; Te = 19.5096
Parâmetros Extraídos (sd = 0.6)	Ampl = 4.5017 Tc = 10.5495; Te = 19.3756
Parâmetros Extraídos (sd = 0.8)	Ampl = 4.6165 Tc = 11.1527; Te = 18.9407
Parâmetros Extraídos (sd = 1)	Ampl = 4.9278 Tc = 12.2870; Te = 20.5936
Parâmetros Extraídos (sd = 1.2)	Ampl = 5.1245 Tc = 12.7501; Te = 20.6820
Parâmetros Extraídos (sd = 1.4)	Ampl = 5.3160 Tc = 12.2129; Te = 18.7089
Parâmetros Extraídos (sd = 1.6)	Ampl = 5.8417 Tc = 14.0937; Te = 21.5737
Parâmetros Extraídos (sd = 1.8)	Ampl = 5.8678 Tc = 12.1748; Te = 16.0908
Parâmetros Extraídos (sd = 2)	Ampl = 6.3978 Tc = 14.1399; Te = 17.0274
Parâmetros Extraídos (sd = 2.2)	Ampl = 6.9649 Tc = 16.1700; Te = 17.1468

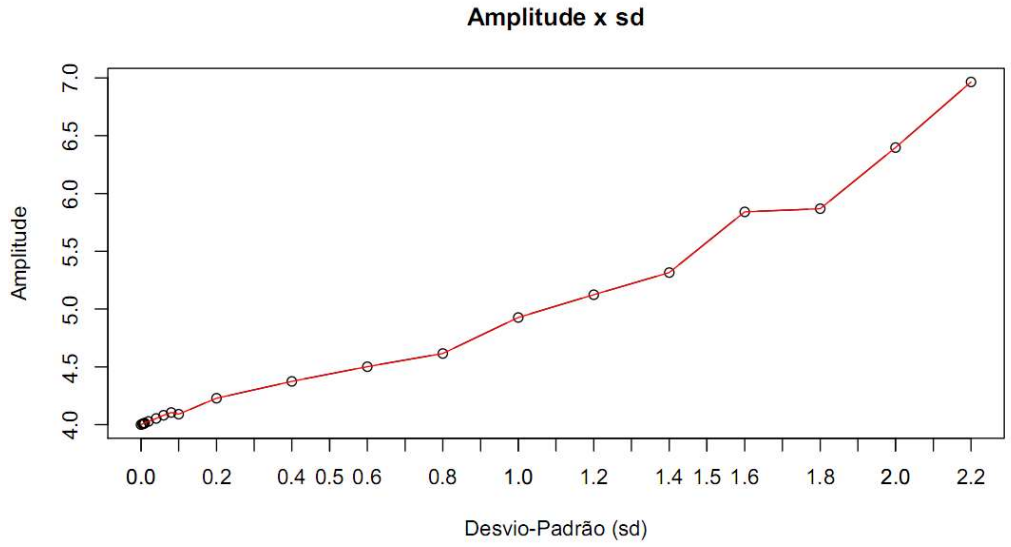
Através dos resultados apresentados na Tabela 4 conclui-se que para um sinal RTS com adição de um ruído térmico, cujo desvio-padrão é equivalente até 10% da amplitude do sinal RTS gerado, o método faz a extração dos parâmetros deste sinal corretamente. Quando o desvio-padrão do ruído térmico atinge o valor de 15% da amplitude do sinal RTS, o valor do parâmetro que fornece a amplitude deste sinal se torna menos preciso, porém os valores de tempo de captura e emissão que são extraídos pelo algoritmo continuam próximos ao valor real. Porém, diferentemente do método A, este método já perde a precisão nos valores de τ_c e τ_e do sinal RTS gerado, quando o desvio-padrão do ruído térmico atinge o valor de 25% da amplitude real do sinal RTS gerado.

Por fim, é possível concluir também que o método se torna incapaz de extrair os valores dos parâmetros do sinal RTS quando o desvio-padrão do ruído térmico adicionado a este sinal atinge um valor de 1.8 (isto é, 45% da amplitude do sinal RTS gerado).

Com base nos resultados mostrados, pode-se concluir que o método B é mais sensível à adição de ruído ao sinal RTS gerado, quando comparado ao método A, pois com valores menores de desvio-padrão já perde sua precisão ao extrair os valores dos parâmetros necessários para caracterizar este sinal.

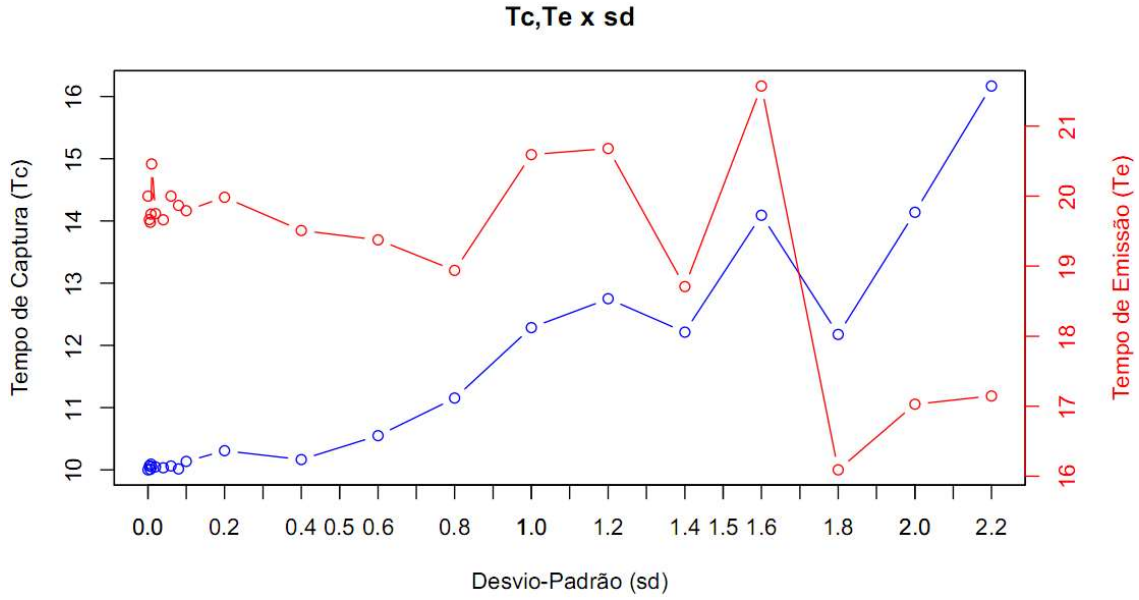
Da mesma forma que para o método anterior, foram construídos gráficos para uma melhor visualização dos resultados da Tabela 4. O gráfico da Figura 21 apresenta uma comparação entre a amplitude do sinal e o desvio-padrão aplicado ao ruído térmico adicionado a este sinal. Enquanto o gráfico da Figura 22 mostra a comparação entre os parâmetros de tempo de captura e tempo de emissão deste sinal com o valor do desvio-padrão utilizado. Com base nos gráficos é possível confirmar como os parâmetros extraídos pelo método se distanciam dos valores gerados, conforme o desvio-padrão do ruído aumenta.

Figura 21 - Resultados da Tabela 4 de Amplitude x sd.



Fonte: Elaborado pela autora.

Figura 22 - Resultados da Tabela 4 de Tc, Te x sd.

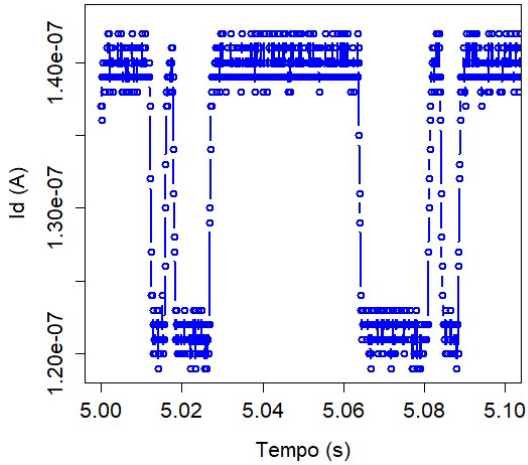


Fonte: Elaborado pela autora.

Por último, ambos os métodos foram testados extraindo os parâmetros de um arquivo de dados experimentais, os quais foram medidos em um dispositivo MOSFET. Os resultados deste teste podem ser observados na Tabela 5. Uma janela de 1000 pontos dos dados

experimentais da corrente ao longo de tempo deste dispositivo pode ser visualizada na Figura 23.

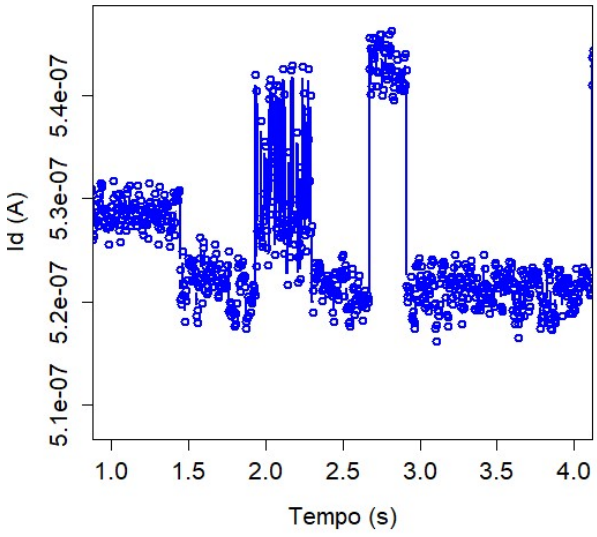
Figura 23 - Curva de Id x Tempo de um dispositivo MOSFET.



Fonte: Elaborado pela autora.

Estes mesmos testes também foram realizados com um arquivo de dados experimentais medidos em um dispositivo ReRAM, os quais estão demonstrados na Tabela 6. Uma janela de 1000 pontos dos dados experimentais da corrente ao longo de tempo deste dispositivo pode ser visualizada na Figura 24.

Figura 24 - Curva de Id x Tempo de um dispositivo ReRAM.



Fonte: Elaborado pela autora.

Assim como acontece com os dados sintéticos gerados anteriormente, é possível notar que os dois métodos são capazes, também, de extrair os parâmetros de medidas experimentais. Então, é possível realizar a caracterização de um RTN através de seus dados medidos em laboratório. É importante ressaltar que para ambos os dispositivos, tecnologias de fabricação diferentes irão gerar resultados diferentes.

Tabela 5 - Resultados da extração de parâmetros a partir de dados experimentais medidos em um dispositivo MOSFET.

Parâmetros Extraídos com o método A	Parâmetros Extraídos com o método B
Ampl = 1.794e-8 A Tc = 11.327e-3 s Te = 13.485e-3 s	Ampl = 1.607e-08 A Tc = 10.941e-3 s Te = 13.845e-3 s

Tabela 6 - Resultados da extração de parâmetros a partir de dados experimentais medidos em um dispositivo ReRAM.

Parâmetros Extraídos com o método A	Parâmetros Extraídos com o método B
Ampl = 1.925e-8 A Tc = 1.325 s Te = 0.395 s	Ampl = 2.277e-08 A Tc = 1.492 s Te = 0.354 s

5 CONCLUSÃO

Neste trabalho foram descritas as características estatísticas do ruído *Random Telegraph Noise* (RTN), a importância da correta caracterização dos parâmetros deste sinal, principalmente em dispositivos eletrônicos de dimensões nanométricas, onde a caracterização estatística adequada pode contribuir com informações essenciais sobre a física destes dispositivos e explicar o papel e as propriedades atomísticas dos defeitos envolvidos na geração do RTN.

Também foram discutidas técnicas de análise, as quais são utilizadas no estudo dos sinais RTN e na recuperação dos parâmetros que caracterizam este sinal. Foram apresentados os métodos mais comuns utilizados, os quais permitem a extração dos parâmetros do sinal RTN a partir de dados medidos, discutindo suas vantagens e desvantagens.

A fim de se atingir o objetivo final do trabalho, foi implementado um método de extração de parâmetros de sinais RTN de dois níveis. Este método baseia-se no algoritmo *Baum-Welch.* Com o desenvolvimento deste algoritmo foi possível realizar a extração de parâmetros de sinais RTN a partir de dados sintéticos e dados experimentais medidos em dispositivos eletrônicos, tais como a ReRAM e o MOSFET.

Além disso, foi apresentado um segundo método de extração de parâmetros, o qual se baseia na discretização de medidas. Consequentemente, foi possível realizar uma comparação entre o método desenvolvido neste trabalho e o segundo método.

Finalmente, uma comparação dos resultados extraídos por cada um dos métodos, possibilitou a realização de uma análise de desempenho de ambos os algoritmos implementados, na presença de ruído Gaussiano. Além disso, também se analisou o desempenho de ambos os métodos na extração de parâmetros de dados experimentais de RTN medidos em MOSFETs e ReRAMs.

Logo, com base nos resultados obtidos conclui-se que ambos os métodos são capazes de extrair corretamente os parâmetros que geraram o sinal RTN em análise. Notou-se ainda que a adição de ruído Gaussiano ao sinal faz com que os métodos percam precisão na extração dos parâmetros do sinal, se afastando dos valores de parâmetros esperados conforme o desvio-padrão do ruído térmico aumenta.

Notou-se também, com base nos resultados, que o método B é mais sensível à adição de ruído ao sinal RTN gerado, quando comparado ao método A, pois este método perde a precisão na extração dos parâmetros com valores menores do desvio padrão do ruído Gaussiano adicionado.

Por fim, com aplicação de ambos os métodos na extração de parâmetros de dados experimentais de RTN medidos em MOSFETs e ReRAMs, conclui-se que ambos foram capazes de realizar a extração dos parâmetros, os quais caracterizam o sinal.

Os métodos apresentados neste trabalho são aplicados em sinais RTN de dois níveis. Sugere-se, portanto, para trabalhos futuros na área a inclusão de técnicas de extração de parâmetros para ruídos RTN de mais de dois níveis. Além disso, é possível ainda realizar o estudo sobre o acoplamento de armadilhas, pois até o momento, consideram-se as armadilhas como sendo mutuamente independentes.

REFERÊNCIAS

BOTH, Thiago Hanna. Autocorrelation analysis in frequency domain as a tool for MOSFET low frequency noise characterization. 2017.

CHURBANOV, Alexander; WINTERS-HILT, Stephen. Implementing EM and Viterbi algorithms for Hidden Markov Model in linear memory. **BMC bioinformatics**, v. 9, n. 1, p. 1-15, 2008.

COGHLAN, Avril. Little book of R for bioinformatics. 2011.

COSTA, Washington César de Almeida et al. Reconhecimento de fala utilizando modelos de Markov escondidos (HMM's) de densidades contínuas. 1994.

DEMPSTER, Arthur P.; LAIRD, Nan M.; RUBIN, Donald B. Maximum likelihood from incomplete data via the EM algorithm. **Journal of the Royal Statistical Society: Series B (Methodological)**, v. 39, n. 1, p. 1-22, 1977.

DE ESTATISTICA, Curso; FARIA, Victor Basılio. Estimaçao de Máxima Verossimilhança via Algoritmo EM, 2011.

DE OLIVEIRA, Luiz Eduardo Soares; MORITA, Marisa Emika. Introduçao aos modelos escondidos de markov (hmm). **2000**, 2000.

ESPINDOLA, Luciana da Silveira. Um Estudo sobre Modelos Ocultos de Markov HMM-Hidden Markov Model. **Porto Alegre, junho de**, 2009.

FORNEY, G. David. The viterbi algorithm. **Proceedings of the IEEE**, v. 61, n. 3, p. 268-278, 1973.

GHAHRAMANI, Zoubin; JORDAN, Michael I. Factorial hidden Markov models. **Machine learning**, v. 29, n. 2, p. 245-273, 1997.

JURAFSKY, Daniel; MARTIN, James H. Speech and language processing (draft). **Chapter A: Hidden Markov Models (Draft of September 11, 2018). Retrieved March**, v. 19, p. 2019, 2018.

KURDTHONGMEE, W. A Modified HMM Forward Algorithm for an Embedded Motion Type Classification. **International Journal o fSignal processing System**, v. 2, n. 2, p. 84-90, 2014.

LANDEIRO, Victor Lemes. Introdução ao uso do programa R. **Manaus: Instituto Nacional de Pesquisas da Amazônia**, 2011.

LOU, H.-L. Implementing the Viterbi algorithm. **IEEE Signal processing magazine**, v. 12, n. 5, p. 42-52, 1995.

MELOS, Ricardo Carvalho de. Trapsimulator: um simulador didático de ruído RTN. 2018.

MOON, Todd K. The expectation-maximization algorithm. **IEEE Signal processing magazine**, v. 13, n. 6, p. 47-60, 1996.

MOR, Bhavya; GARHWAL, Sunita; KUMAR, Ajay. A systematic review of hidden markov models and their applications. **Archives of Computational Methods in Engineering**, p. 1-20, 2020.

NILSSON, Mikael. **First order hidden markov model: Theory and implementation issues**. 2005.

PRAKASH, Amit; HWANG, Hyunsang. Multilevel cell storage and resistance variability in resistive random access memory. **Physical Sciences Reviews**, v. 1, n. 6, 2016.

PUGLISI, Francesco M. et al. RTS noise characterization of HfOx RRAM in high resistive state. **Solid-State Electronics**, v. 84, p. 160-166, 2013a.

PUGLISI, Francesco Maria; PAVAN, Paolo. RTN analysis with FHMM as a tool for multi-trap characterization in HfO x RRAM. In: **2013 IEEE International Conference of Electron Devices and Solid-State Circuits**. IEEE, 2013b. p. 1-2.

PUGLISI, Francesco Maria; PAVAN, Paolo. Factorial hidden Markov model analysis of random telegraph noise in resistive random access memories. **ECTI Transactions on Electrical Engineering, Electronics, and Communications**, v. 12, n. 1, p. 24-29, 2014.

PUGLISI, Francesco Maria et al. Characterization of anomalous random telegraph noise in resistive random access memory. In: **2015 45th European Solid State Device Research Conference (ESSDERC)**. IEEE, 2015. p. 270-273.

PUGLISI, Francesco Maria; PAVAN, Paolo. Guidelines for a reliable analysis of random telegraph noise in electronic devices. **IEEE Transactions on Instrumentation and Measurement**, v. 65, n. 6, p. 1435-1442, 2016.

PUGLISI, Francesco Maria et al. Random telegraph noise: Measurement, data analysis, and interpretation. In: **2017 IEEE 24th International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)**. IEEE, 2017. p. 1-9.

PUGLISI, Francesco Maria et al. Random telegraph noise in resistive random access memories: Compact modeling and advanced circuit design. **IEEE Transactions on Electron Devices**, v. 65, n. 7, p. 2964-2972, 2018.

PUGLISI, F. M. Noise in Resistive Random Access Memory Devices. In: **Noise in Nanoscale Semiconductor Devices**. Springer, Cham, 2020. p. 87-133.

RABINER, Lawrence R. A tutorial on hidden Markov models and selected applications in speech recognition. **Proceedings of the IEEE**, v. 77, n. 2, p. 257-286, 1989.

RAMESH, Sridharan. HMMs and the forward-backward algorithm.—2010. **URL: http://people. csail. mit. edu/rameshvs/content/hmms. pdf**.

RITTER, Matias do Nascimento; THEY, Ng Haig; KONZEN, Enéas Ricardo. Introdução ao software estatístico R. 2019.

SEO, Jaehyun et al. Automatic ReRAM SPICE model generation from empirical data for fast ReRAM-circuit coevaluation. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 25, n. 6, p. 1821-1830, 2017.

SILVA, Maurício Banaszeski da. A physics-based statistical random telegraph noise model. 2016.

TATAVARTY, Usha Ramya. Implementation of numerically stable hidden Markov model. 2011.

TENYAKOV, Anton. Estimation of hidden Markov models and their applications in finance. 2014.