Tiago Sanches da Silva

Plataforma de estudo para determinação de conectividade cerebral embarcada e em tempo real

Brasil

São Paulo, 2016

Tiago Sanches da Silva

Plataforma de estudo para determinação de conectividade cerebral embarcada e em tempo real

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências.

Universidade de São Paulo – USP

Escola Politécnica - Departamento de Telecomunicações e Controle

Programa de Pós-Graduação

Orientador: Prof. Dr. Luiz Antonio Baccalá

Brasil

São Paulo, 2016

Silva, Tiago

Plataforma de estudo para determinação de conectividade cerebral embarcada e em tempo real/ T. Silva – São Paulo, 2016.

118 p.

Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Telecomunicações e Controle.

1.Coerência Parcial Direcionada 2.Algoritmo Embarcado 3.Processamento em Tempo Real 4.Conectividade Cerebral 5.Medicina I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Telecomunicações e Controle II.t.

Este trabalho é dedicado primeiramente a Deus que até aqui me guiou,
a minha esposa Jessyka e minha familia pelo apoio em todos os momentos.

Agradecimentos

Agradeço primeiramente à Deus por não me deixar desistir e nem me levar por sentimentos de derrota e cansaço. Também quero agradecer à minha esposa, meus pais e irmãos que me apoiaram com suas orações, cuidados e palavras de ânimo. Amo muito todos vocês.

Agradeço também aos meus amigos que, do jeito deles, sempre me apoiaram em tudo, não somente no mestrado. Especialmente aos amigos do coração e dos campos da justiça: Igor Yanes , Gyorgy Laszlo, Igor Valério e Rodrigo Dobbin.

Um agradecimento especial ao Prof. Luiz Baccalá pela oportunidade de fazer mestrado e me aceitar como seu orientando. E que ao compartilhar comigo um pouco de sua experiência, me ajudou a crescer como profissional e como pessoa. Muito obrigado, professor, por me dar suporte para finalizar meu mestrado nessa fase turbulenta da vida.

Agradeço também à minha atual empresa, LG Eletronics, por me ceder dias para que eu trabalhasse na dissertação do meu mestrado.

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

A presente dissertação examina um método de determinação da conectividade cerebral cujo uso vem se tornando popular nos últimos anos, o *partial direct coherence* (PDC), que se destaca dentre outros métodos por possibilitar a verificação das relações imediatas de sinais multivariados. Este método representa a conectividade cerebral no domínio da frequência e tem íntima relação com a noção de "causalidade" de Granger (GRANGER, 1969), que possibilita quantificar a influência mútua entre séries temporais observadas.

De um ponto de vista computacional, o referido método faz uso de modelos de séries temporais que hoje têm implementação bastante eficiente em termos de algoritmos *off-line*, mas cujo sucesso depende da presunção de estacionariedade dos dados, fato que é somente verdadeiro em trechos relativamente curtos de sinais de origem cerebral, como no caso do EEG (Eletroencefalograma).

O objetivo deste trabalho é criar um sistema que calcule o PDC, continuamente, em tempo real e que possua a mesma precisão do método *off-line*, além de ser uma plataforma de estudos para implementações e testes de métodos de determinação da conectividade neural em tempo real.

A plataforma desenvolvida é modular, incentivando futuros trabalhos na mesma, e mostrou-se eficaz quanto a precisão numérica dos resultados do cálculo do PDC. As características de tempo real foram atingidas com algumas restrições, que dependem da configuração do usuário e do número de canais que um sinal possui.

Palavras-chave: coerência parcial direcionada. algoritmo embarcado. processamento em tempo real. conectividade cerebral. medicina.

Abstract

This thesis examines a method of determination of brain connectivity whose use becomes popular in recent years, the partial direct coherence (PDC) that stands out in comparison with other methods for making possible the verification of immediate relations of multivariate signal. This method represents the brain connectivity in the frequency domain and has a close relationship with the notion of Granger causality (GRANGER, 1969) that makes it possible to quantify the mutual influence between observed time series.

From a computational perspective, the above method makes use of time series models, which today has very efficient implementation in terms of off-line algorithm, but whose success depends on presume that the data is stationary, a fact that is only true in relatively short stretches of cerebral signals, especially in the case of EEG.

The objective of this thesis is to create a system that calculates the PDC continuously and in real time maintaining the same precision of the off-line method. Furthermore being a research platform for implementations and tests of new methods for determining neural connectivity in real time.

The developed platform is modular encouraging future work on it, and was effective in the numerical accuracy of the PDC calculation results. The real time characteristics were achieved with some restrictions that depend of the user configuration and the number of channels that the signal has.

Keywords: partial direct coherence. embedded algorithms. real time processing. brain connectivity. medicine.

Lista de ilustrações

Lista de tabelas

Lista de símbolos

σ	Matriz de covariância
λ	Frequência normalizada
π	Matriz de coerência parcial direcionada
$\hat{\mathbf{\Delta}}_p$	Coeficientes de Reflexão

Sumário

Parte I

Introdução

Introdução

Na última década se viu um interesse crescente em métodos de estimação de conectividade cerebral. Essa é uma ideia recente em que motor é a necessidade de busca de um novo paradigma de investigação que permite entender os detalhes de funcionamento do cérebro. Essa ideia vem em adição a abordagens sistemáticas que permearam os anos 90 e 2000 e cuja ênfase era a de investigar o diferencial aumento de atividade cerebral em diferentes contextos de estímulo (LI, 2013). Durante esse período, as principais modalidades de sinal foram essencialmente a PET (Positron Emission tomography) (MYERS; CUNNINGHAM; BAILEY, 1996; PHELPS, 2006) e a chamada ressonância magnética funcional (FMRI - Functional Magnetic Ressonance Imaging) (STIPPICH; SARTOR, 2007; STROMAN, 2011), que se caracterizam por permitir a avaliação indireta da atividade neural por meio de variações de consumo de metabólito. Por sorte, as regiões de respostas mais intensas podem ser detectadas, relativamente, devido ao consumo cerebral médio. No todo, este tipo de abordagem se tornou conhecido como "Neo Frenologia", ao permitir delimitar quais regiões respondem a determinados estímulos, em um paralelo a ideias em voga no Sec. XIX e, que como agora, se revelaram pouco esclarecedoras dos mecanismos neurobiológicos seja em cognição ou em análise comportamental (COOTER, 1984).

Em contraponto a esse cenário, a partir do início dos anos 90 (KAMINSKI; BLINOWSKA, 1991) começou-se a considerar abordagens alternativas que permitissem estudar as interrelações entre regiões cerebrais. Uma breve visão dessa evolução pode ser apreciada em (SAMESHIMA; BACCALá, 2014). A maioria desses métodos se deu no contexto de processamento de sinais de EEG (Eletroencefalografia) e representa uma evolução de técnicas espectrais que vinham sendo aplicadas a pares de sinais provenientes de eletrodos intracranianos ou de escalpo (FREEMAN; QUIROGA, 2012; MITRA; BOKIL, 2008; TONG; THAKOR, 2009). Desde cedo, ao contrário de técnicas neofrenológicas (POLDRACK; MUMFORD; NICHOLS, 2011), esse novo tipo de abordagem enfrentou o problema de caracterização estatística de seu desempenho que só recentemente conheceu solução rigorosa (BACCALA et al., 2013).

Dentre as técnicas propostas, a chamada coerência parcial direcionada ou PDC, do acrônimo em inglês de *partial directed coherence* (BACCALA; SAMESHIMA, 2001b) teve destaque por permitir a resolução das relações imediatas entre estruturas no caso de análise de sinais multivariados (BACCALA; SAMESHIMA, 2001a). A avaliação da PDC e de outras técnicas tem sido feita preponderantemente com base em processamento *off-line* e há pouca informação sobre a viabilidade prática dessas técnicas no contexto de processamento em tempo real.

Por que processamento em tempo real? O motivo é simples: o sinal de EEG, além de barato, é de fácil coleta e sua análise clínica é, em geral, realizada por pessoal médico treinado no reconhecimento de padrões que surgem em diversos contextos clínicos, que vão de análise de estágios de sono (LEE-CHIONG, 2008) até a epilepsia (VARSAVSKY; MAREELS; COOK, 2010), em que determinar o chamado foco epiléptico tem papel crucial em casos refratários ao tratamento medicamentoso (FOUNDATION, 2002).

Em palavras mais simples e objetivas: o corpo clínico está acostumado a realizar diagnóstico e interpretar os dados a medida em que são observados, já que correlatos comportamentais como tremores e perda de atenção podem ser imediatamente levados em conta junto com as evidências neuroelétricas. Isso significa que métodos que procurem revelar estados neurofisiológicos precisam estar sujeitos a análise no instante em que o ensaio é realizado. Sem isso, qualquer tentativa de transladar essas técnicas para o contexto da aplicação clínico-diagnóstico está fadada ao insucesso.

Um exame breve da metodologia envolvida na determinação da PDC basta para elegê-la como candidata à implementação em tempo real como se discute no Cap. 1.

O objetivo da presente dissertação é estudar a viabilidade prática dessa proposta no contexto da PDC.

O restante desse trabalho se completa pela discussão de aspectos de uma proposta de implementação (Cap. 2), seguida de sua caracterização quanto a desempenho e eficiência (Cap. 3). O trabalho termina com um balanço e sugestões para trabalhos futuros (Cap. 4).

Parte II

Formulação do problema

1 Introdução teórica

No presente capítulo provê-se uma breve recapitulação do problema de determinação de conectividade cerebral, iniciando pelos conceitos mais básicos e mostrando que o problema pode ser expresso em termos de modelos de análise de séries temporais (BROCKWELL; DAVIS, 1987; LüTKEPOHL, 2005) cuja metodologia de estimação se presta a implementações embarcadas.

1.1 Causalidade e Conectividade Cerebral

Em 1969, Clive Granger propôs uma maneira de avaliar o efeito prático que a evolução de uma variável de interesse exerce sobre a outra (GRANGER, 1969). Tal conceito recebeu o nome de *Causalidade de Granger* e pode ser resumida da seguinte forma: Uma série temporal $x_1(n)$ "Granger-cause"outra série temporal $x_2(n)$, se o conhecimento do passado de $x_1(n)$ melhora a capacidade de prever o presente de $x_2(n)$. Uma forma de medir esta melhoria é através do erro quadrático médio de predição:

$$E\left[|x_2(n) - \hat{x}_{2|21-}|^2\right] << E\left[|x_2(n) - \hat{x}_{2|2-}|^2\right] \tag{1.1}$$

na qual o símbolo com circunflexo indica o estimador de $x_2(n)$ sendo, respectivamente, $_{|2-}$ baseado em seu próprio passado contra $_{|21-}$ que inclui o passado de $x_1(n)$, este segundo como melhor estimador.

No contexto clássico de séries temporais, o melhor preditor linear em senso amplo de um processo $x_2(n)$, pode ser obtido ajustando um modelo regressão com base no seu próprio passado:

$$x_2(n) = \sum_{r=1}^{p} a_r x_2(n-r) + w(n) \tag{1.2}$$

na qual o coeficiente a_r representa a influência que as observações r-atrasos no passado possuem sobre a observação atual. A Eq. (1.2) descreve o sinal $x_2(n)$ como processo autorregressivo (BROCKWELL; DAVIS, 1987) de ordem p, enquanto $w(n)$ representa as chamadas "inovações", isto é, coleta toda informação não contida no passado de $x_2(n)$. Um modelo autorregressivo (AR(p)) de ordem p é adequado quando $w(n)$ for essencialmente "branco".

Os coeficientes a_r de (1.2) podem ser facilmente obtidos por uma abordagem de minimização quadrática, o qual implica que a variância de $w(n)$ representa o erro

$E\left[|x_2(n) - \hat{x}_{2|2-}|^2\right]$ em que

$$\hat{x}_{2|2-} = \sum_{r=1}^{p} a_r x_2(n-r) \tag{1.3}$$

é o preditor linear de interesse.

Generalizando esta ideia, em que o passado de $x_1(n)$ também está disponível, conduz a

$$\hat{x}_{2|21-} = \sum_{r=1}^{p} a_r x_2(n-r) + \sum_{r=1}^{p} b_r x_1(n-r) \tag{1.4}$$

tal que, se o seu erro $E\left[|x_2(n) - \hat{x}_{2|21-}|^2\right]$ satisfizer (1.1), seguirá a propriedade de causalidade de Granger. Uma maneira equivalente de verificar essa condição é testar a hipótese estatística se $b_r = 0$, em que se rejeita a causalidade.

Esta formulação mostra claramente ser possível que $x_1(n)$ "Granger-cause"$x_2(n)$ sem que $x_2(n)$ "Granger-cause"$x_1(n)$. Esta ausência de reciprocidade permite a interpretação de uma direcionalidade no fluxo de informação e forma a base do estudo de conectividade cerebral.

Essas ideias podem ser generalizadas para um contexto com mais de duas séries temporais, algo que na realidade é imprescindível (BACCALA; SAMESHIMA, 2001a). Para tanto, se reformula essa forma de modelamento coletando o sinal de interesse num vetor $\mathbf{x}(n) = [x_1(n) \ \ldots x_N(n)]^T$ que obedeça ao modelo autorregressivo vetorial de ordem p:

$$\begin{bmatrix} x_1(n) \\ \vdots \\ x_N(n) \end{bmatrix} = \sum_r \mathbf{A}_r \begin{bmatrix} x_1(n-r) \\ \vdots \\ x_N(n-r) \end{bmatrix} + \begin{bmatrix} w_1(n) \\ \vdots \\ w_N(n) \end{bmatrix} \tag{1.5}$$

em que as influências passadas da série x_j, devidas ao instante $n-r$ sobre $x_i(n)$ são representadas pelos coeficientes $a_{ij}(r)$ da matriz

$$\mathbf{A}_r = \begin{bmatrix} a_{11}(r) & a_{12}(r) & \ldots & \ldots & a_{1N}(r) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & a_{ij}(r) & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1}(r) & \ldots & \ldots & \ldots & a_{NN}(r) \end{bmatrix}. \tag{1.6}$$

Nesse caso, pode-se mostrar que a ausência da causalidade de Granger de x_j sobre x_i equivale a hipótese de que

$$a_{ij}(r) = 0, \quad \forall r = 1, \ldots, p \tag{1.7}$$

para dadas observações. De forma semelhante ao caso do modelo (1.2), cujos coeficientes podem ser obtidos por mínimos quadrados (BROCKWELL; DAVIS, 1987), o mesmo vale para modelos VAR(p) (LüTKEPOHL, 2005). Como veremos a seguir no tópico1.3 a estimação dos modelos pode ser feita de maneira eficiente.

1.2 Coerência Parcial Direcionada

Em neurociência, especialmente em EEG, a representação dos sinais no domínio da frequência tem uma interpretação fisiológica importante (BASAR, 2004; BUZSÁKI, 2006), no sentido de capturar ideia de causalidade de Granger medida pelo conjunto dos coeficientes $a_{ij}(r)$, em (BACCALA; SAMESHIMA, 2001b), definiu-se a chamada coerência parcial direcionada na frequência normalizada $\lambda \in [-0.5, 0.5]$:

$$\pi_{ij}(\lambda) = \frac{\bar{A}_{ij}(\lambda)}{\sqrt{\sum_{k=1}^{N} |\bar{A}_{kj}(\lambda)|^2}} \tag{1.8}$$

na qual $\bar{A}_{ij}(\lambda)$ é o elemento i, j da matriz:

$$\bar{A}_{ij}(\lambda) = \begin{cases} 1 - \sum\limits_{r=1}^{p} a_{ij}(r) e^{-\mathbf{j}2\pi\lambda r}, \text{ se } \ i = j \\ - \sum\limits_{r=1}^{p} a_{ij}(r) e^{-\mathbf{j}2\pi\lambda r}, \text{ do contrário} \end{cases} \tag{1.9}$$

de modo que a condição de ausência de causalidade (1.7) equivale a

$$|\pi_{ij}(\lambda)|^2 = 0 \tag{1.10}$$

cujos aspectos de inferência podem ser apreciados em (TAKAHASHI; A.; SAMESHIMA, 2007).

A noção de coerência parcial direcionada teve diversas evoluções (TAKAHASHI; BACCALA; SAMESHIMA, 2010; SAMESHIMA; BACCALA, 2014) cujos aspectos de inferência são tratados em (BACCALA et al., 2013).

Por questões de simplicidade e, pelo fato da virtual equivalência inferencial entre as diversas expressões da PDC, neste trabalho somente (1.8) é considerada.

De crucial, porém, é a necessidade de estimação precisa do modelo (1.5) revisto a seguir.

1.3 Algoritmos de Estimação de Modelo

Há diversas maneiras de determinar $a_{ij}(r)$. Todas são, de uma forma ou outra, variantes da minimização de algum parâmetro apropriado. Em geral esta determinação se faz num contexto em que os sinais envolvidos são estacionários em senso amplo e, se faz uso, direta ou indiretamente, de funções de correlação entre os dados observados.

A forma canônica da solução envolve a reescrita de (1.5) na forma de equações de Yule-Walker multivariadas (WHITTLE, 1963; LüTKEPOHL, 2005), conduzindo à necessidade de inversão de uma matriz $Np \times Np$ para cada possível ordem da tentativa.

Os algoritmos podem ser classificados como indiretos, que fazem uso da referida estimativa de correlação, e diretos em que os dados são iterativamente incorporados na obtenção dos coeficientes. Algoritmos deste último tipo, que permitem recursividade de ordem p, são atraentes por terem um número total de operações aritméticas reduzido. Isto é atraente porque a maioria dos dados práticos possuem modelos com p desconhecido '*à priori*' e que, porém, podem ser determinados com o auxílio dos chamados critério de ordem (AKAIKE, 1974; BURNHAM; ANDERSON, 2003; MCQUARRIE; TSAI, 1998). Por simplicidade, daqui em diante vamos presumir que os modelos de interesse possuem uma ordem p_{max} máxima definida.

Outra classificação aplicável a esses algoritmos é: de "batelada"ou "adaptativos"(HAYKIN, 2011), cuja diferença reside em, respectivamente, modelos que são calculados com base em dados que perfazem L amostras temporais por canal, e modelos em que a estimativa é atualizada a cada instante. A literatura descreve esta última abordagem a uma outra aparentada à PDC (MOLLER et al., 2001).

No caso adaptativo, os algoritmos exigem um parâmetro a mais chamado fator de esquecimento, que introduz o equivalente a uma janela efetiva L_{eff}, sendo equivalente a uma janela L de um algoritmo de batelada, caso o fator de esquecimento seja devidamente escolhido. Esse fator impacta em duas propriedades importantes das estimativas: sua variância residual e a velocidade de convergência.

Devido a experiência prática disponível, optou-se aqui por implementar um algoritmo em batelada descrito em (NUTTALL, 1976), conforme a implementação proposta por (MARPLE, 1987), que será recapitulada em mais detalhes no tópico 1.3.1. Esta escolha foi, em parte, motivada pela possibilidade recente da generalização desse algoritmo para aplicações em contexto de interesse psicofísico (RODRIGUES; BACCALA, 2015).

1.3.1 Algoritmo de Nutall-Strand

O algoritmo de Nuttall-Strand (MARPLE, 1987; NUTTALL, 1976) se constitui na generalização multivariada do método recursivo proposto por Burg para estimar modelos

AR(p). Parte-se da ideia explícita de que os sinais envolvidos são localmente estacionários.

Obviamente os valores preditos com base no passado $\hat{\mathbf{x}}_p$ são dados por

$$\hat{\mathbf{x}}_p^f(n) = \sum_{l=1}^{p} \mathbf{A}_p(l)\mathbf{x}(n-l) \tag{1.11}$$

uma vez que a ordem é definida.

Da estacionariedade do processo $\mathbf{x}(n)$, é possível conceber a possibilidade de estimar o valor do processo com base em valores futuros, dando lugar a:

$$\hat{\mathbf{x}}_p^b(n) = \sum_{l=1}^{p} \mathbf{B}_p(l)\mathbf{x}(n+l), \tag{1.12}$$

As Eqs. (1.11,1.12) permitem escrever os erros de predição

$$\mathbf{e}_p^f(n) = \mathbf{x}(n) - \hat{\mathbf{x}}_p^f(n) \tag{1.13}$$

$$\mathbf{e}_p^b(n) = \mathbf{x}(n) - \hat{\mathbf{x}}_p^b(n) \tag{1.14}$$

correspondentes da sorte de que, se $\mathbf{\Sigma}_p^b$ e $\mathbf{\Sigma}_p^f$ forem as matrizes de covariância desses erros, procura-se o melhor modelo de modo a minimizar uma versão devidamente pesada dessas covariâncias:

$$\text{trace}\left(\left(\mathbf{\Sigma}_{p-1}^f\right)^{-1}\hat{\mathbf{\Sigma}}_p^f + \left(\mathbf{\Sigma}_{p-1}^b\right)^{-1}\hat{\mathbf{\Sigma}}_p^b\right), \tag{1.15}$$

que sujeito as relações de atualização dos erros

$$\mathbf{e}_p^f(n) = \mathbf{e}_{p-1}^f(n) - \mathbf{\Delta}_p(\mathbf{\Sigma}_p^b)^{-1}\mathbf{e}_p^b(n-1) \tag{1.16}$$

$$\mathbf{e}_p^b(n) = \mathbf{e}_{p-1}^b(n) - \mathbf{\Delta}_p^H(\mathbf{\Sigma}_p^f)^{-1}\mathbf{e}_p^f(n-1) \tag{1.17}$$

sob a mudança de ordem, em que $\hat{\mathbf{\Delta}}_p$ representa os chamados coeficientes de reflexão, que permitem atualizar os erros do modelo de ordem p com base nos erros do modelo da ordem anterior.

Isto conduz a

$$\hat{\mathbf{\Sigma}}_p^f\left(\mathbf{\Sigma}_p^f\right)^{-1}\hat{\mathbf{\Delta}}_{p+1} + \hat{\mathbf{\Delta}}_{p+1}\left(\mathbf{\Sigma}_p^b\right)^{-1}\hat{\mathbf{\Sigma}}_p^b = -2\hat{\mathbf{\Sigma}}_p^{fb}, \tag{1.18}$$

na qual o circunflexo representa valores estimados a partir dos dados e a quantia $\mathbf{\Sigma}_p^{fb}$ representa a matriz de covariância cruzada entre $\mathbf{e}_p^f(n)$ e $\mathbf{e}_p^b(n)$.

A eq. (1.18) é um caso particular de equação matricial

$$\mathbf{AX} + \mathbf{XB} = \mathbf{C} \tag{1.19}$$

para o qual existem diversas formas econômicas e precisas de solução (MARPLE, 1987; DUAN, 2015). Porém, por simplicidade, neste trabalho optou-se pela abordagem mais ingênua, que consiste em resolver (1.19) lançando mão de propriedades do produto de Kronecker (LüTKEPOHL, 1996) em que vale

$$\text{vec}(\mathbf{ABC}) = \mathbf{C}^T \otimes \mathbf{A}\text{vec}(\mathbf{B}) \tag{1.20}$$

na qual vec$(\cdot)$ é o operador que reorganiza as colunas de uma matriz, empilhando-as sequencialmente num vetor. Sua aplicação a (1.19) produz

$$\left[\mathbf{I} \otimes \mathbf{A} + \mathbf{B}^T \otimes \mathbf{I}\right] \text{vec}(\mathbf{X}) = \text{vec}(\mathbf{C}) \tag{1.21}$$

ou seja

$$\text{vec}(\mathbf{X}) = \left[\mathbf{I} \otimes \mathbf{A} + \mathbf{B}^T \otimes \mathbf{I}\right]^{-1} \text{vec}(\mathbf{C}) \tag{1.22}$$

cujo uso em (1.18) permite calcular $\hat{\mathbf{\Delta}}_{p+1}$ e dela obter a matriz associada a nova ordem por meio de (MARPLE, 1987)

$$\mathbf{A}_{p+1}(p+1) = \hat{\mathbf{\Delta}}_{p+1}\mathbf{\Sigma}_p^f \tag{1.23}$$

Nesse contexto, daqui em diante essa solução será referida como "*naive*", ou ingênua, sendo a escolhida no presente desenvolvimento embarcado.

A menos deste último passo, de solucionar (1.18) pelo método ingênuo, a implementação segue as linhas contidas no pacote **AsympPDC** disponível em (SAMESHIMA; BACCALá, 2014) e que pode ser baixado do site <http://www.lcs.poli.usp.br/$\sim$baccala/pdc>. O referido pacote resolve (1.18) com uma rotina nativa do *toolbox* de controle do MatLab.

1.3.1.1 Cálculo Rápido da PDC

O cálculo propriamente dito da PDC passa a obter os valores contidos em (1.9) para cada valor de frequência λ de interesse. O cálculo direto por substituição é ineficiente, mas existe, uma alternativa que parte da noção de interpolação no domínio da frequência por adição de zeros (*"zero padding"*) (PERCIVAL; WALDEN, 1993).

Sucintamente, o procedimento consiste em tomar a sequência:

$$[\delta_{ij} \ -a_{ij}(1) \ -a_{ij}(2) \ \ldots \ -a_{ij}(p)] \tag{1.24}$$

na qual δ_{ij} é o delta de Kronecker, e completá-la com zeros

$$[\delta_{ij} \ -a_{ij}(1) \ -a_{ij}(2) \ \ldots \ -a_{ij}(p) \ 0 \ \ldots \ 0] \tag{1.25}$$

de modo que o comprimento total de (1.25) seja o dobro do número de pontos $(2 * L)$ desejados no domínio da frequência. É conveniente escolher este comprimento como o

potências de base 2, pois a aplicação da FFT ao (1.25) resulta em uma outra sequência complexa cujos L primeiros pontos são os valores desejados de $\bar{A}_{ij}(\lambda)$ equiespaçados no intervalo $\lambda \in [0, .5]$.

Parte III

Implementação

2 Implementação

As tecnologias utilizadas neste trabalho foram escolhidas de modo a tornar possível criar um sistema de tempo real, com alta mobilidade, fácil integração com outros sistemas e portabilidade para futuras implementações e melhorias. O sistema criado deve realizar análises e cálculos de maneira contínua, gerando saídas em intervalos definidos pelo usuário.

A definição de tempo real para essa aplicação é na ordem de segundos, e deve ser definido pelo usuário. Dado os tipos de sistemas de tempo real existentes, o RealTimeMcarns encaixa-se no de *soft real time* não crítico, ou seja, mesmo que exista algum atraso na resposta do sistema em relação ao tempo especificado, isso não irá acarretar em risco para o usuário ou ao sistema e ainda existe valor na informação gerada.

A atual implementação possui uma restrição de intervalo mínimo entre os ciclos de cálculo em função de parâmetros, como características do sinal de entrada e ordem do algoritmo que estima o modelo autorregressivo multivariado, Nuttall-Strand. Essa restrição é devido a performance da solução atual e será apresentado com maiores detalhes no capitulo Testes e Resultados.

2.1 Linguagem de programação

Para este trabalho escolheu-se criar uma solução completa utilizando a linguagem de programação C, uma linguagem estruturada e de uso geral, com compiladores disponíveis para diversos microcontroladores e microprocessadores existentes no mercado.

A linguagem C é portável entre arquiteturas computacionais distintas, o que possibilita a alta portabilidade para a solução. Sem maiores dificuldades, é possível compilar o código para qualquer Sistema Operacional (SO) e também para plataformas Bare Metal, ou seja, software embarcado que é executado sem auxílio de um SO.

Os compiladores C tendem a gerar código eficiente, visto que a linguagem possui estruturas que são mapeadas diretamente em instruções de máquina.

2.2 Cenários considerados

Existem soluções em MatLab que realizam os cálculos de maneira *off-line*, ou seja, é necessário realizar a coleta de um conjunto de dados em um laboratório, levar para um computador que possua uma licença do MatLab e executar os algoritmos com os dados coletados. Essas soluções que dependem de um modelo autoregressivo geralmente precisam resolver equações do tipo Lyapunov-Sylvester, e o MatLab utiliza uma biblioteca auxiliar

chamada SLICOT (*Subroutine Library in Systems and Control Theory*) para solucionar tais equações.

A biblioteca de sub-rotinas SLICOT fornece implementações eficientes, em Fortran 77, de algoritmos numéricos para sistemas computacionais e teoria de controle. Ela é baseada nas bibliotecas numéricas de álgebra linear BLAS (*Basic Linear Algebra Subprograms*) e LAPACK (*Linear Algebra Packege*), também escritas em Fortran. Por essa razão considerou-se criar um sistema capaz de realizar os cálculos de coerência parcial direcionada (PDC) de modo contínuo em um computador com alta capacidade de processamento, além da possibilidade de utilizar bibliotecas como a SLICOT para facilitar a implementação do sistema. Porém, ter uma plataforma com mobilidade abre novas oportunidades para ensaios em locais desfavorecidos de infraestrutura, e isto foi uma fator mais determinante que o desempenho e a facilidade na implementação.

As bibliotecas SLICOT, BLAS e LAPACK podem ser baixadas gratuitamente através dos sites:<http://slicot.org/> , <http://www.netlib.org/blas/> e <http://www.netlib.org/lapack/> acessados em janeiro de 2016.

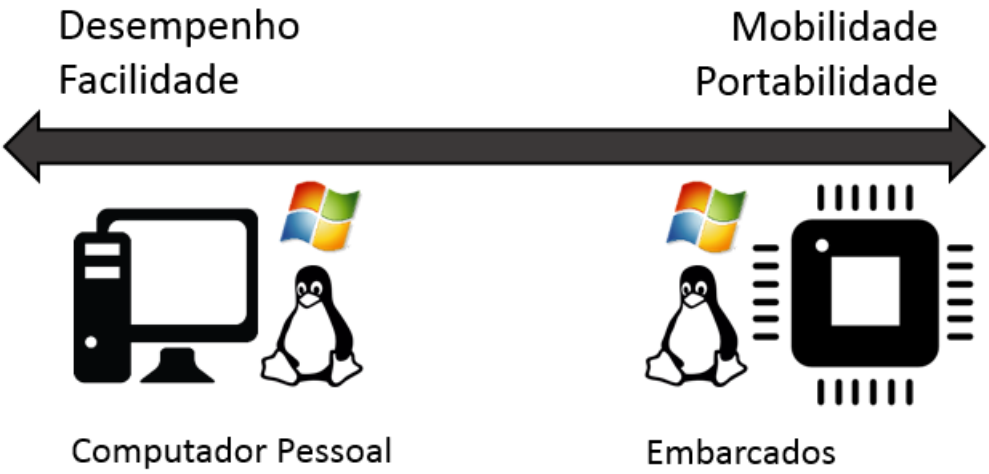

Figura 1: Computador x Embarcados

Uma plataforma móvel pode incentivar a comunidade científica a utilizar o sistema desenvolvido, dado que o mesmo torna viável realizar ensaios em ambientes que antes não seriam possíveis.

Considerando a importância da mobilidade, a plataforma embarcada foi escolhida para o desenvolvimento do projeto, e junto com esta escolha surgiram alguns desafios como escrever uma biblioteca de álgebra linear totalmente em C, para que seja portável para qualquer plataforma; atingir os requisitos de tempo real e criar um sistema capaz de realizar a coleta de dados e executar os cálculos de maneira continua.

Entende-se que existem muitas opções quanto a implementação: computador uti-

lizando uma placa de vídeo para auxiliar nos cálculos matriciais, uma FPGA (*Field Programmable Gate Array*), uma placa de desenvolvimento com DSP (*Digital Signal Processor*) embarcado entre muitas outras possibilidades. Porém lembre-se que está é uma plataforma para estudo de algoritmos de conectividade em tempo real, e que visa oferecer as ferramentas necessárias para esse estudo. O objetivo não é criar uma plataforma definitiva para a solução, mas sim criar uma que possibilite o estudo através da manipulação e modificação dos algoritmos de maneira simples e fácil, bem como portar a solução para outro hardware sem muitos problemas de compatibilidade. E através desses estudos e melhorias um dia chegar a uma implementação no estado de arte para conectividade cerebral em tempo real.

2.3 Placa de desenvolvimento

No mercado brasileiro existem várias placas de desenvolvimento com alto desempenho que serviriam para esse projeto. Optou-se por utilizar a Wand Board Quad, pois além da facilidade em adquiri-la, a mesma era uma das melhores do mercado (novembro de 2014), na Figura 2 pode-se observar uma imagem da placa.

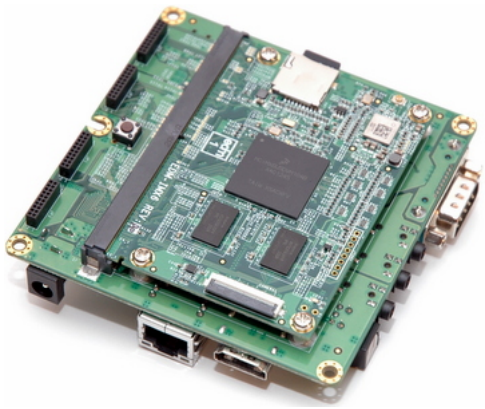

Figura 2: Placa de desenvolvimento - Wand Board Quad

Uma característica que também influenciou na escolha da placa foi o fato dela possuir um processador baseado na arquitetura ARM (*Advanced RISC Machine*). Inúmeros fabricantes constroem seus processadores utilizando essa arquitetura, aumentando as possibilidades de uma possível migração futura do sistema, sem a necessidade de adequação e nem mesmo recompilação, se mantido o mesmo sistema operacional. ARM é uma empresa criada em Cambridge, e que não fabrica chips, apenas propriedade intelectual para arquitetura de processadores e microcontroladores. Esta placa possui um processador ARM – Cortex A9 Quad Core, 2GB de RAM, *Float Point Unit* (FPU) entre outras características, na Tabela 1 encontram-se as especificações dessa placa de desenvolvimento.

Tabela 1: Características de hardware da placa Wand Board Quad

Pocessador	Freescale i.MX6 Quad 1GHz
Núcleos	Cortex-A9 Quad Core
Gráficos	Vivante GC 2000
	+ Vivante GC 355
	+ Vivante GC 320
Memória	2GB DDR3
Audio	Sim
HDMI	Sim
Slots SD-Card	2
Porta seria	Sim
USB	2
SATA	1
Gigabit Ethernet	Sim
Wifi	Sim
Bluetooth	Sim
I/Os de propósitos gerais	4x20

Fonte: <http://www.wandboard.org/details>

Essa arquitetura teve uma rápida aceitação do mercado devido ao seu alto desempenho e robustez. Grandes fabricantes como Texas Instruments, Freescale, NXP, Atmel, Toshiba, Fujitsu Semicondutor entre outros, oferecem soluções baseadas em ARM. Esta placa possui também uma placa gráfica embarcada que pode ser utilizada em trabalhos futuros a fim de melhorar o desempenho de alguns cálculos matriciais custosos como multiplicação de matrizes.

2.4 Sistema operacional

O sistema operacional escolhido para compor a solução foi o Linux, considerando seu baixo custo, o suporte da comunidade e a quantidade de distribuições criadas para uma mesma plataforma, além da experiência anterior do desenvolvedor. A distribuição escolhida foi uma versão enxuta do Ubuntu 14.04 para ARM, com suporte a FPU implementado em hardware.

2.5 Visão geral do sistema

O nome da plataforma desenvolvida neste trabalho é RealTimeMcarns e será chamado a partir deste momento de RTMcarns, apenas por simplificação.

Considerando que esta é uma primeira implementação e existem muitos aspectos em que o projeto pode ser melhorado, como citado anteriormente, um dos objetivos deste trabalho é servir como uma plataforma base para estudos de algoritmos de tempo real em

conectividade cerebral. Então a solução deve ser modular, para que possa evoluir de forma fácil e simples, incentivando futuros trabalhos na plataforma projetada.

Na Figura 3 pode ser observado o diagrama de blocos simplificado do sistema. Cada um desses módulos será visto detalhadamente em tópicos posteriores.

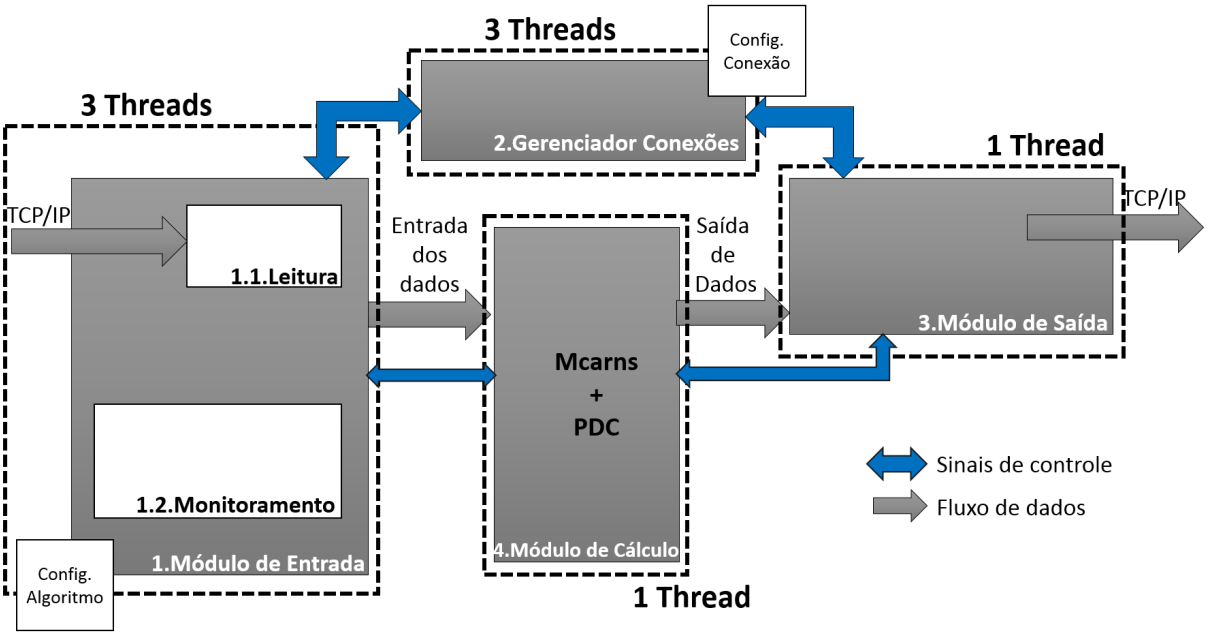

Figura 3: Visão geral do RealTimeMcarns

O sistema foi dividido em quatro grandes módulos: Módulo de Entrada, Gerenciador de Conexões, Módulo de Saída e Módulo de Cálculo.

Resumidamente, pode-se dizer que o Módulo de Entrada é responsável por realizar a leitura dos dados que chegam pela porta TCP (*Transmission Control Protocol*), além de interpretar, validar e montar a matriz de entrada em modo contínuo. Esse módulo também é responsável por monitorar os parâmetros do sistema, como **quando** a Entrada deve enviar os dados para o Módulo de Cálculo dando início a um ciclo de cálculo.

O Módulo de Cálculo contém os algoritmos que estimam os modelos de regressão linear através do método de Nuttall-Strand e também a PDC. Ainda neste módulo controla-se quais resultados devem ser gerados além de quais serão armazenados em disco local. Os arquivos criados pelo RTMcarns são do tipo *comma-separated values* (csv), em português: valores separados por um delimitador. Esses arquivos possuem as matrizes dos modelos de regressão linear do sinal de entrada geradas pelo método de Nuttall-Strand, e também as matrizes PDC resultantes desses modelos. Esses arquivos foram muito utilizados na fase de validação dos algoritmos, pois esses arquivos são facilmente importados pelo MatLab, facilitando a verificação e comparação de resultados.

Os resultados gerados pelo Módulo de Cálculo são enviados para o Módulo de Saída,

que os serializa e empacota para que possam ser corretamente interpretados pelo cliente que irá recebe-los. O envio dos resultados também é realizado via porta TCP. Cliente será o nome dado neste trabalho a qualquer dispositivo que se conecta a uma porta TCP do RTMCanrs.

O Gerenciador de Conexão monitora as portas TCP utilizadas para entrada e saída do sistema, que por padrão são respectivamente 17000 e 17001, mas podem ser modificadas pelo usuário via um arquivo de configuração de conexão. Este módulo monta dois servidores TCP que aguardam por conexões de clientes.

Os Módulos de Entrada e Saída ficam pausados enquanto não existe conexão, e ao ser estabelecida, o Gerenciador de Conexões envia sinais para que os módulos sejam restaurados e iniciem a comunicação com cliente TCP conectado.

2.6 Entradas e saídas do sistema

A placa de desenvolvimento possui muitas interfaces para comunicação: USB (*Universal Serial Bus*), SATA (*Serial AT Attachment*), *Ethernet*, Wi-Fi, Bluetooth e 20 pinos de propósitos gerais. Porém escolher USB, SATA ou até mesmo Bluetooth implicaria que na restrição dos dispositivos conectados na entrada e na saída estivessem fisicamente próximos do sistema. Então optou-se por definir um protocolo e não uma interface como padrão de comunicação do RTMcarns.

A opção de escolher a pilha TCP/IP como protocolo padrão de comunicação do sistema se deu para que não exista uma forte dependência de uma interface de comunicação. No contexto do projeto, qualquer uma das interfaces pode ser redirecionada para a porta TCP do localhost (endereço IP local do dispositivo). As interfaces Ethernet e Wi-Fi não precisam de nenhum software auxiliar, basta que o sistema esteja conectado em uma rede para que todos os dispositivos nela possam ter acesso remoto ao RealTimeMcarns.

2.6.1 TCP/IP

Em meados de 1973, um grupo chamado CERF, que trabalhava com rede de pesquisas Stanford, trabalhou no desenvolvimento inicial do protocolo TCP/IP. Posteriormente, a empresa DARPA foi contratada pelas universidades Stanford e College London para desenvolver algumas versões do TCP-IP incluindo a TCP/IP v4 que utilizamos até hoje.

O protocolo TCP/IP é padronizado e todos os sistemas operacionais o suportam. É robusto, possui escalabilidade, é multiplataforma e possui estrutura cliente/servidor, além de possibilitar o acesso direto a internet.

Caso exista a necessidade de utilizar alguma interface local específica, basta construir um software que funcionará como ponte interpretando dados da interface desejada e

redirecionando para a porta TCP local. Tomando como exemplo a USB, deve-se ler os dados recebidos da USB e reenviá-los para a porta TCP de entrada do RealTimeMcarns no endereço IP local. Todo sistema funcionará perfeitamente sem que exista a necessidade de alteração do código. Exemplo ilustrado na Figura 4.

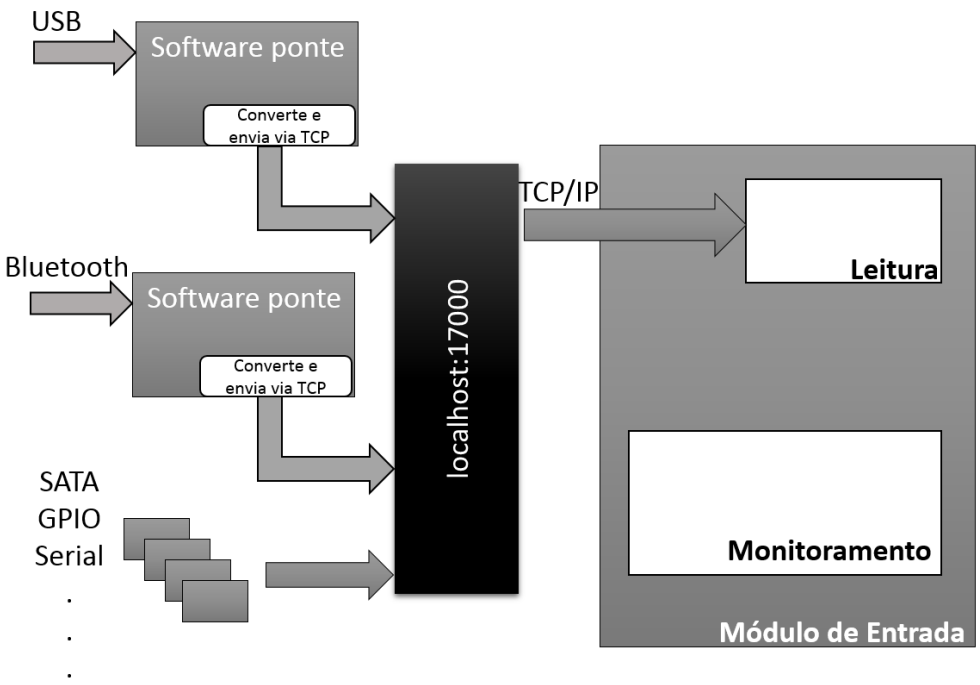

Figura 4: Ilustração do redirecionamento da comunicação para uma porta TCP

Com essa implementação baseada em TCP/IP é possível enviar os resultados de ensaios para qualquer lugar do mundo através da internet, basta que o endereço e porta estejam acessíveis para a rede externa.

2.6.2 TCP x UDP

O protocolo e transporte UDP também oferece algumas características citadas no tópico anterior, além disso um dos maiores motivos para lembrar-se dele é o baixo *overhead* quando comparado com o TCP. *Overhead* é uma expressão utilizada para indicar o excesso de transmissões que o protocolo TCP utiliza, pois ele requisita a confirmação de entrega de um pacote, além do tamanho do cabeçalho TCP ser maior por ele realizar controle de fluxo, de erros e retransmissão.

O protocolo UDP (*User Datagram Protocol*) é menos complexo, mais enxuto, mas que não existe garantia na entrega do pacote e nem da sequência de entrega dos dados.

TCP é uma solução confiável para ambientes não confiáveis. Com ele você tem a garantia da entrega do fluxo de bytes fim a fim. O TCP foi projetado para se adaptar

dinamicamente às propriedades da rede (TANENBAUM, 2011).

No contexto deste trabalho, o valor de cada informação enviada à entrada do sistema é alto e a perda de uma sequência de amostras irá culminar em resultados errôneos, por consequência, na baixa confiabilidade do sistema. O mesmo ocorrerá caso os dados cheguem fora de ordem. Dentre todas as características do TCP a mais importante é a integridade garantida dos dados fim a fim. E por isso ele foi a escolha como protocolo de transporte desse trabalho.

2.7 Entendendo o funcionamento do sistema

Na Figura 5 pode-se observar, de uma maneira geral, como o sistema interage com dispositivos externos e como é o fluxo contínuo de informação.

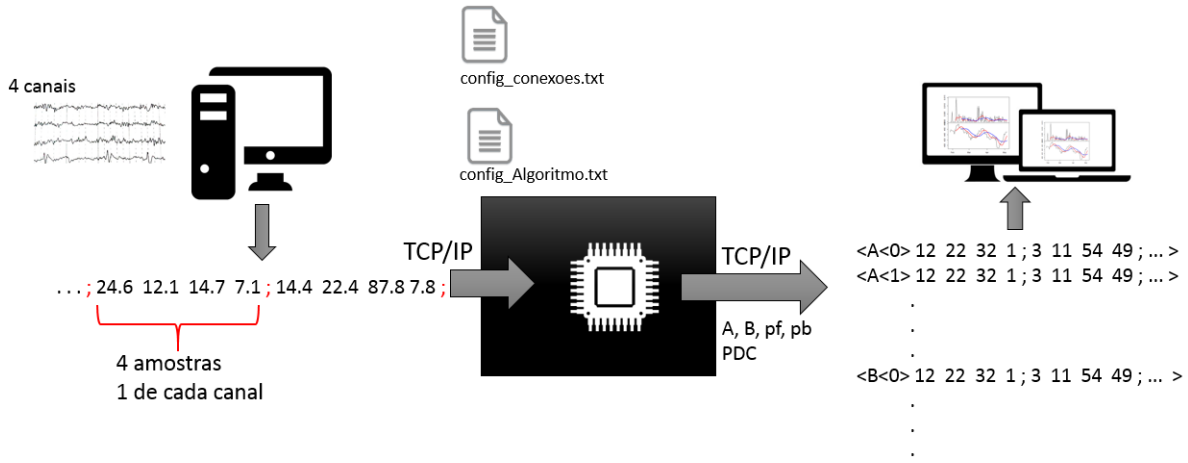

Figura 5: Visão geral do funcionamento continuo do sistema

Em resumo, o usuário irá conectar ao RTMcarns qualquer dispositivo capaz de enviar dados via TCP/IP. A porta padrão para conexão na entrada do sistema é 17000 e na de saída é a 17001. E com outro dispositivo ele deverá conectar-se na porta de saída para receber os resultados gerados e exibi-los de forma gráfica. O dispositivo conectado na entrada deve enviar as series de dados com uma amostra de cada canal intercaladamente, como será mostrado no próximo tópico.

2.7.1 Quanto à entrada

O dispositivo conectado na entrada deve realizar a conversão de analógico para digital do sinal, serializa-los e enviar para o RTMcarns com uma amostra de cada canal seguido do delimitador “;” (ponto e vírgula), como mostrado na Figura 6.

O delimitador ';' serve apenas para sincronismo e identificação de erro. O sistema monitora se entre os delimitadores existem sempre o mesmo número de amostras. Isso

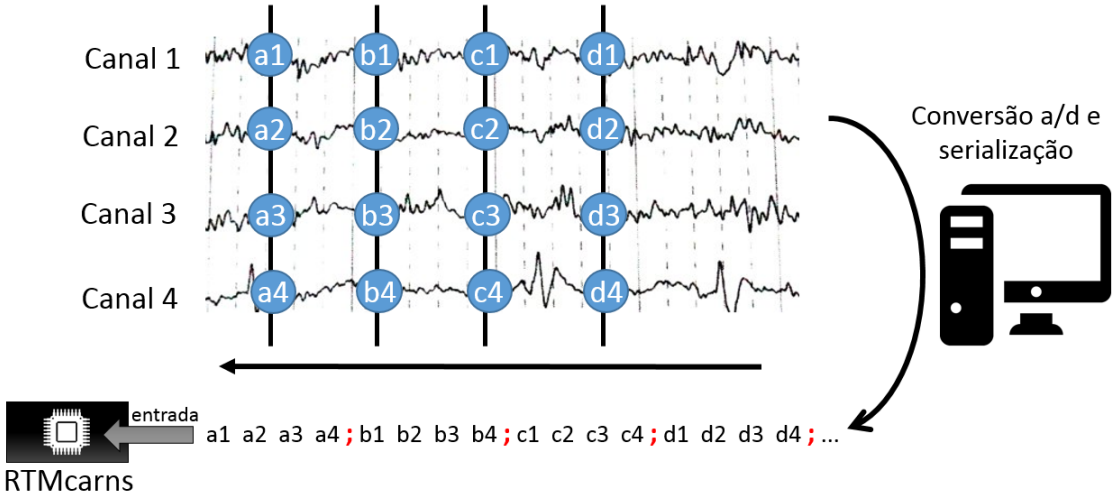

Figura 6: Formato que os valores do sinal de entrada devem ser enviados para o RTMcarns

previne o desalinhamento da matriz de entrada que é construída com os dados que chegam pela TCP, pois a cada ';' é verificado se os valores de cada canal estão presentes na amostragem e, caso falte o valor de algum canal o conjunto todo é descartado.

Existe um arquivo de configuração do sistema chamado 'config_Algoritmo.txt', em que o usuário deve informar o número de canais que tem o sinal que será enviado para o sistema, utilizando essa informação o RTMcarns confere cada conjunto de dados na entrada.

Os valores de cada amostragem devem ser transformados em uma cadeia de caracteres para serem enviadas ao RTMcarns via TCP, por exemplo para representar o valor 8,2315 deve ser enviada uma cadeia de seis caracteres "8,2315". Entre os valores de cada canal deve existir um caractere ' ' (espaço), salvo quando finalizado com o caractere de sincronismo ';'.

2.7.2 Quanto à saída

Existem duas maneiras de se obter os resultados do sistema: A primeira é conectando um dispositivo na porta TCP de saída do RTMcarns e a segunda é através dos arquivos csv que são salvos para cada matriz gerada e em cada ciclo de cálculo. Esses arquivos são armazenados localmente em uma pasta chamada *out*, como será mostrado em tópicos posteriores.

2.7.2.1 Via TCP

No arquivo 'config_Algoritmo.txt', o usuário poderá escolher se deseja gerar arquivos locais csv dos resultados e das entradas que as geraram. Também poderá definir

quais resultados serão enviados para a saída TCP a cada ciclo de cálculo. As matrizes que podem ser enviadas para a saída são:

- $\mathbf{A}_p$: Matrizes de coeficiente de predição linear para frente.
  - Identificadas no sistema como 'A'.
  - Enviadas via TCP com identificador 'A' (ID=A).
  - Salvo localmente no arquivo como 'A'.
- Matriz de covariância do erro da predição linear para frente.
  - Identificadas no sistema como 'pf'.
  - Enviadas via TCP com identificador 'W' (ID=W).
  - Salvo localmente no arquivo como 'pf'.
- $\mathbf{B}_p$: Matrizes de coeficientes de predição linear para trás.
  - Identificadas no sistema como 'B'.
  - Enviadas via TCP com identificador 'B' (ID=B).
  - Salvo localmente no arquivo como 'B'.
- Matriz de covariância do erro da predição linear para trás.
  - Identificadas no sistema como 'pb'.
  - Enviadas via TCP com identificador 'X' (ID=X).
  - Salvo localmente no arquivo como 'pb'.
- $\pi_{ij}(\lambda)$: Matriz de coerência parcial direcionada.
  - Identificadas no sistema como 'PDC'.
  - Enviadas via TCP com identificador 'F' (ID=F).
  - Salvo localmente no arquivo como 'PDC'.

O número de canais do sinal de entrada será chamado de N_c e a ordem do modelo de regressão linear de p.

Cada ciclo de cálculo resulta em p matrizes A e B, uma matriz de f, pb e PDC. Sendo que a matriz PDC é tridimensional com $N_c \times N_c \times 128$ elementos, sendo a terceira dimensão referente ao número de raias no domínio da frequência para o cálculo do PDC, onde dos 128 são: p matrizes referentes aos coeficientes de $\mathbf{A}_p$ e o restante de *padding* para a execução do FFT (*Fast Fourier Transform*).

As matrizes de saída possuem o seguinte formato:

$$< \mathbf{ID} < \mathbf{índice} < v_{11}\ v_{12}\ v_{13}\ v_{14}\ ;\ v_{21}\ v_{22}\ v_{23}\ v_{24}\ ;\ v_{31}\ v_{32}\ v_{33}\ v_{34}\ ;\ v_{41}\ v_{42}\ v_{43}\ v_{44} >$$

onde, **ID** $\in [A, W, B, X]$, **índice** $\in [0 : (p-1)]$ e os valores enviados são de uma matriz 4x4.

A matriz PDC (**ID**=F) por ser tridimensional é enviada de uma forma diferente, dado F_{ijk}, os vetores formados nas posições i e j são enviados separadamente:

$$< \mathrm{F} < \mathrm{i}\ ,\ \mathrm{j} < v_1\ v_2\ v_3\ v_4\ v_5\ \ldots\ v_{128} >$$

onde, $i \in [0 : (N_c - 1)]$ e $j \in [0 : (N_c - 1)]$.

Cada um dos valores das matrizes são transformados em uma cadeia de caracteres para serem enviados pela TCP, dessa forma o dispositivo ligado a saída do sistema deve ser capaz de ler a cadeia de caracteres e transformá-los em valores numéricos.

Na Figura 7 pode ser observado uma exemplificação das saídas geradas.

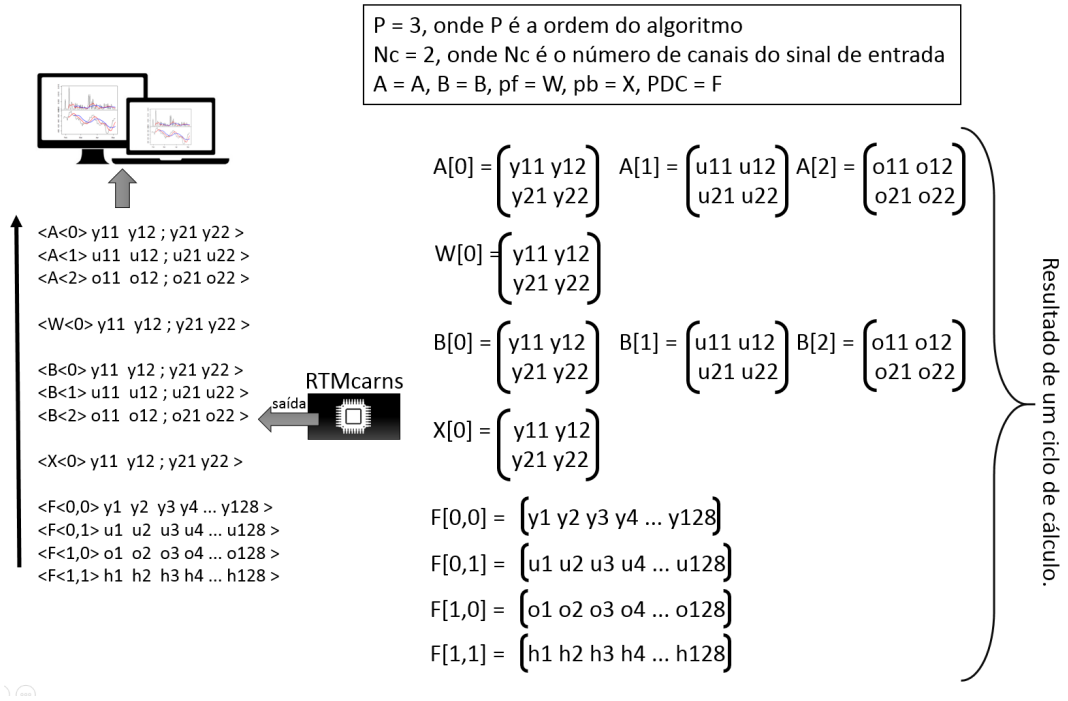

Figura 7: Ilustração de como as matrizes existentes na memória do sistema são serializadas e enviadas via TCP

O RTMcarns, até o presente momento deste trabalho, não gera interface gráfica alguma para apresentar os resultados ao usuário. O dispositivo conectado à saída deve interpretar os resultados enviados via TCP e criar uma interface conveniente à área de aplicação.

2.7.2.2 Armazenamento local

Caso o sistema esteja configurado para salvar as entradas e saídas em disco local, diversas matrizes serão armazenadas a cada ciclo de cálculo.

No início da execução do sistema são criadas duas pastas: *in* e *out*. Na pasta *in* são armazenadas as matrizes de entrada utilizadas no cálculo. Na pasta *out* as matrizes resultantes do cálculo.

Como exemplo podemos verificar na Figura 8 a estrutura de pastas criadas após a execução do RTMcarns. Neste exemplo o sistema foi configurado para realizar os cálculos utilizando modelos de regressão de ordem 3, e foram realizados dois ciclos de cálculo em sequência.

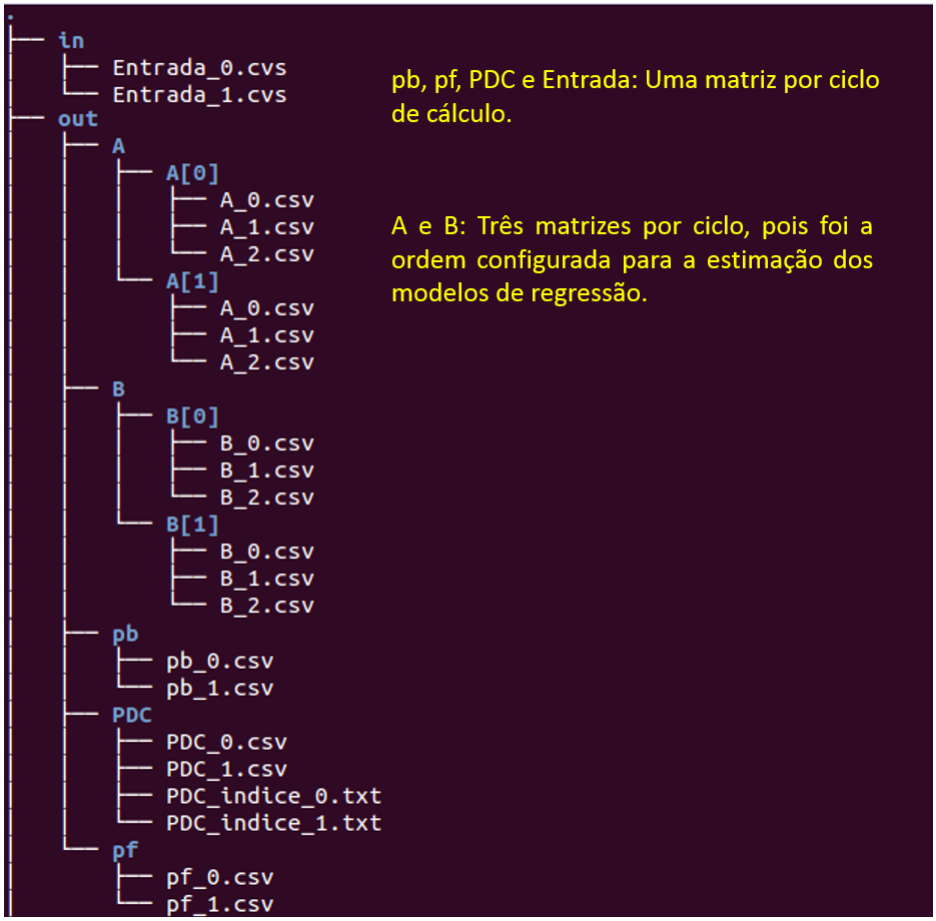

Figura 8: Imagem do console utilizando o comando *tree* após a execução do RTMcarns com duas series de dados seguidas e $p = 3$

Após a execução de cada ciclo de cálculo são gerados arquivos csv das matrizes de entrada e saída, veja abaixo a lista de matrizes armazenadas para cada ciclo:

- Uma matriz de entrada.
- Uma matriz de covariância do erro da predição linear para frente (pf).

- Uma matriz de covariância do erro da predição linear para trás (pb).
- Uma matriz de coerência parcial direcionada (PDC).
- Uma pasta A[ciclo] que contém p matrizes de coeficientes autoregressivos.
  - Onde p é a ordem do modelo de regressão escolhido pelo usuário e ciclo é o número do ciclo atual desde a inicialização do sistema, sendo o primeiro ciclo = 0.
- Uma pasta B[ciclo] que contém p matrizes de coeficientes autoregressivos.
  - Onde p é a ordem do modelo de regressão escolhido pelo usuário e ciclo é o número do ciclo atual desde a inicialização do sistema, sendo o primeiro ciclo = 0.

Dado que PDC é uma matriz tridimensional, ela é escrita no arquivo csv de forma que todos os vetores da terceira dimensão fossem serializados de cima pra baixo. Para que o usuário possa reconstruí-lo corretamente em algum software matemático como o Matlab, é criado juntamente com o arquivo csv um txt de índices. Observe Figura 9 a relação entre o arquivo 'PDC_0.csv' e 'PDC_indice_0.txt'.

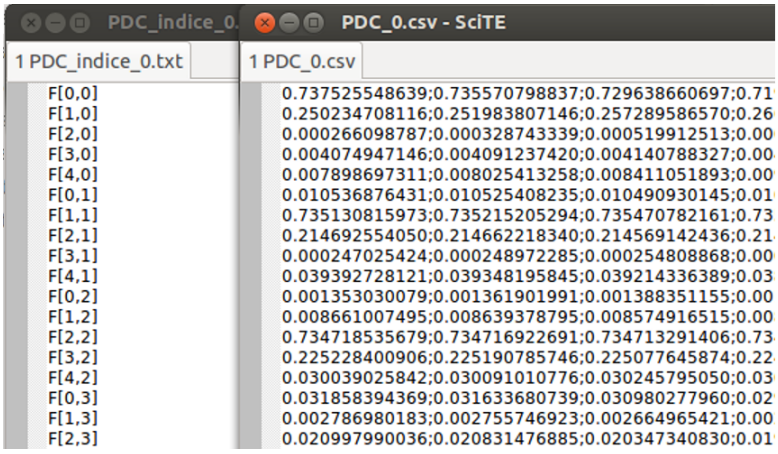

Figura 9: A esquerda o arquivo 'PDC_indice_0.txt' exibe qual o índice (i,j) respectivo da matriz tridimensional que pertence o vetor que está no arquivo 'PDC_indice_0.txt', assim possibilitando a correta reconstrucao da matriz tridimensional PDC

2.7.3 Software auxiliar de envio de dados

Para incentivar o uso do sistema como plataforma de estudos, foi criado o *software* Teste_envio, uma aplicação auxiliar que elimina a necessidade de um sistema real de coleta de dados, como um equipamento de eletroencefalograma.

Esse *software* realiza a leitura de arquivos csv e serializa os dados no formato correto para o RTMcarns. Para que ele encontre os arquivos deve existir uma pasta local

chamada 'envio' e os arquivos dentro dela devem ser nomeados da seguinte forma: u0.csv, u1.csv, u2.csv até u999.csv. Esse é um *software* simples que, basicamente, envia dados de maneira contínua para RTMcarns a partir dos arquivos lidos na pasta 'envio'.

Os argumentos de entrada do Teste_Envio são o IP e a porta TCP de entrada configurada no RTMcarns. Após conectado ele inicia o envio contínuo dos dados para o RTMcarns, o intervalo de envio entre as amostragens pode ser configurado no arquivo 'ConfigEnvio.txt'. Veja na Figura 10 a ilustração do funcionamento do sistema utilizando o software auxiliar.

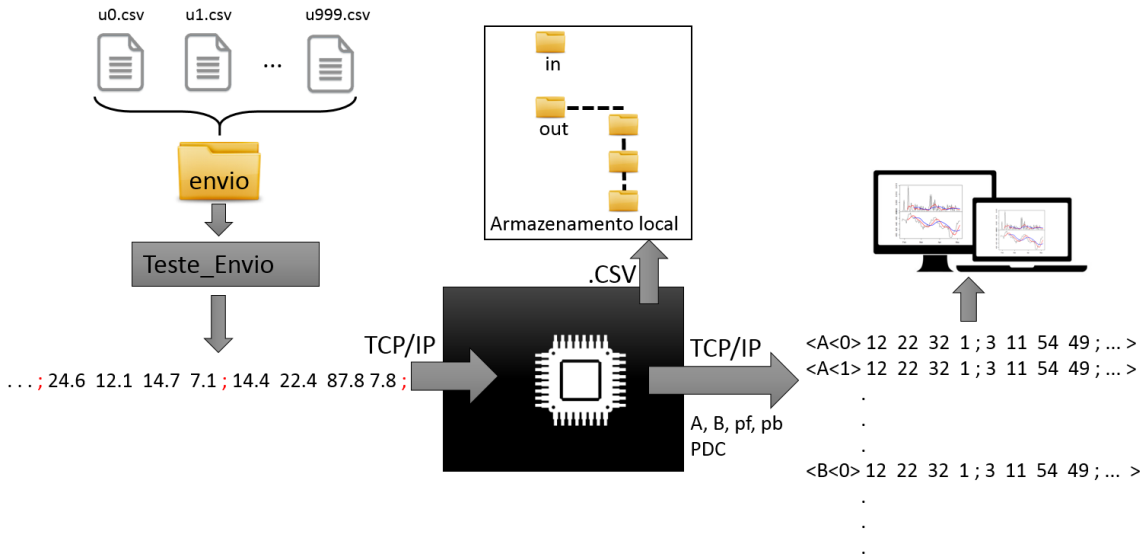

Figura 10: Visão geral do sistema com a utilização do *software* Teste_Envio

No arquivo "ConfigEnvio.txt" deve ser configurado basicamente o número de canais do sinal que será enviado e o intervalo entre as amostragens em microssegundos. É esperado que esse cenário incentive o trabalho na plataforma criada, sendo que elimina a necessidade de equipamentos complexos e outros sistemas de difícil acesso. O Teste_Envio está disponível para a plataforma Linux, e pode ser executado de qualquer máquina virtual com acesso a mesma rede em que o RTMcarns estiver conectado.

2.8 Configuração do sistema

Para que o RTMcarns opere da forma esperada pelo usuário, é necessário aprender como configurá-lo e entender como cada parâmetro interfere no funcionamento do mesmo. Existem dois arquivos para configurar o sistema: 'config_Algoritmo.txt' e 'config_conexoes.txt'.

O arquivo 'config_conexoes.txt' é utilizado para configurar as portas TCP utilizadas pelo sistema.

```
'Porta TCP de Entrada'
E=17000
'Porta TCP de Saída'
S=17001
```

Essas portas foram definidas após consulta ao IANA (*Internet Assigned Numbers Authority*), que é a organização mundial com máxima autoridade na atribuição dos números das portas na Internet. E essas duas portas estão disponíveis para propósitos gerais. A lista completa pode ser consultada em no site do IANA <http://www.iana.org/assignments/service-names-port-numbers>, acessado em janeiro de 2016.

O arquivo "config_Algoritmo.txt" possui mais campos que configuram o comportamento do algoritmo de cálculo. Os valores que estão preenchidos em cada um dos campos que serão apresentados são apenas para exemplificação. Os dois primeiros campos são: quantidade de canais e ordem do algoritmo.

```
'Qtd. Canais.'
L=4
'Ordem do modelo de regressão'
Z=12
```

A quantidade de canais do sinal que será recebido pelo RTMcarns deve ser informada, para que o Módulo de Entrada possa identificar caso ocorra perda de informação. A ordem é utilizada para configurar o algoritmo que calcula os coeficientes autoregressivos do modelo de regressão pelo método Nutall-Strand.

Os próximos seis campos definem se as saídas e entradas devem ser armazenadas localmente e quais as saídas intermediárias devem ser geradas. O resultado final PDC é sempre gerado pelo RTMcarns.

```
'Armazenar as Saídas localmente? (S-Sim , N-Não )'
R=S
'Armazenar as Entradas localmente? (S-Sim , N-Não )'
Y=S
'Selecionar quais as Saídas intermediárias deverão ser geradas pelo Algoritmo
'A'
V=S
'B'
Q=S
'pf'
W=S
```

```
'pb'
X=S
```

Os próximos cinco parâmetros influenciam no modo de operar do sistema, bem como os a interação do Módulo de Entrada com o Módulo de Cálculo.

```
'Intervalo mínimo entre as saídas (em segundos)'
T=4
'Intervalo máximo entre as saídas (em segundos).'
I=20
'Número de amostras novas mínimas para efetuar o cálculo.'
N=800
'Número de amostras que deverão ser mantidas no vetor para envio ao Módulo de
M=2000
'Qual delas terá prioridade?'
'Intervalo máximo: O=i'
'Número de amostras: O=n'
O=n
```

O esperado é que o sistema receba continuamente dados pela entrada, e esses parâmetros acima possibilitam configurar o momento em que eles devem ser enviados para o Módulo de Cálculo.

O parâmetro 'T' define o intervalo mínimo, em segundos, entre os ciclos de cálculo do sistema, independente de quantas amostras tenham sido recebidas na entrada.

Após passar o tempo configurado em "T", caso o número mínimo de amostras novas 'N' tenham sido atingidas, a matriz de entrada é criada e enviada para o Módulo de Cálculo. Caso o número mínimo de amostras ainda não tenha sido atingido, o sistema aguarda até que o tempo decorrido seja igual ao configurado no campo 'I', intervalo máximo entre as saídas.

Ao passar o tempo máximo de intervalo configurado, e o número mínimo de amostras novas ainda não for atingido, então o campo 'O' é consultado. Ele define qual parâmetro tem prioridade no comportamento do sistema. Se 'O=i', então o cálculo irá iniciar quando o intervalo máximo tiver decorrido do último ciclo. Caso 'O=n' o número mínimo de amostras novas serão aguardadas antes que se inicie o ciclo de cálculo, não importando quanto tempo isso demore. A Figura 11 mostra um fluxograma simplificado dessa lógica.

No momento da criação da matriz de entrada para o envio ao Módulo de Cálculo, existe a possibilidade de utilizar amostras que já foram enviadas em ciclos anteriores de

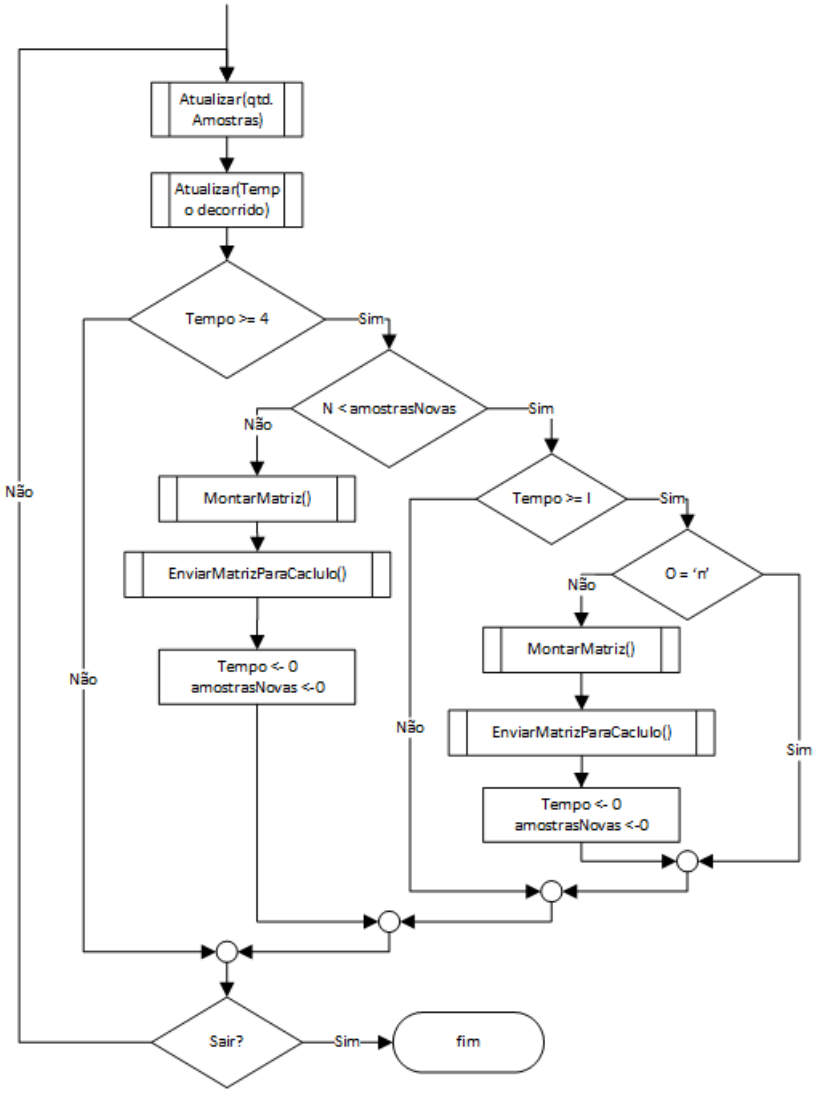

Figura 11: Fluxograma simplificado da lógica do comportamento do sistema segundo a configuração do usuário

cálculo. Basta que o parâmetro 'M' seja maior que o 'N'. Tomando como exemplo os valores atuais, 'N=800' e 'M=2000', significa que a matriz de entrada que será enviada para o Módulo de Cálculo possui 2000 amostras, em que 800 são novas e 1200 de ciclos anteriores. Esse recurso é interessante quando o usuário deseja inserir nos resultados uma noção de continuidade, sendo que ele terá sobreposição de amostras na entrada entre os resultados dos ciclos.

Se esta sobreposição for indesejada, basta inserir um valor para "M" que seja igual a "N". Dessa maneira, somente as amostras novas serão utilizadas para montar a matriz de entrada. Esse comportamento será abordado com mais detalhes em tópicos posteriores.

2.9 Sistema *multithread*

O sistema RTMcarns é *multithread*, sendo assim os módulos são executados paralelamente entre si.

Thread é um encadeamento de execução, uma forma de dividir um processo e executar duas ou mais tarefas de maneira concorrente. Resumidamente, programação *multithread* é a técnica que permite projetar e implementar aplicações paralelas de maneira eficiente. O desenvolvimento desse tipo de aplicação não é simples, devido a um conjunto de problemas intrínsecos como a comunicação e sincronismo entre os *threads*, concorrência de recursos e condições de corrida.

Neste trabalho utilizou-se regiões compartilhadas de memória para comunicação entre *threads*, protegidas por semáforos mutualmente exclusivos (mutex), que impedem o acesso simultâneo a um recurso. Os semáforos são inicializados com herança de prioridade ativo para minimizar possíveis problemas com inversão de prioridade de uma *thread*.

O sincronismo é feito com variáveis condicionais que esperam por um determinado sinal para remover a *thread* do estado bloqueada. O suporte à *thread* é fornecido pelo próprio sistema operacional, sendo assim existe uma dificuldade em criar um projeto multiplataforma que utilize programação *multithread*. Por este motivo optou-se, neste trabalho, por utilizar POSIX *Threads* (pthreads), uma API (*Application Programming Interface*) padrão que pode ser encontrado em diversos sistemas operacionais.

2.9.1 POSIX *Threads*

POSIX é o acrônimo para *Portable Operationg System Interface*, e trata-se uma família de normas bem definidas pelo IEEE (atextitInstitute of Electrical and Electronics Engineers) para a compatibilidade entre sistemas operacionais. Designada formalmente por IEEE 1003, ela define uma API, para compatibilidade de aplicações entre sistemas operacionais distintos.

Ao utilizar bibliotecas POSIX, a compatibilidade de código fonte entre sistemas operacionais que atendem as normas é garantida. Todos os SOs derivados do UNIX são compatíveis nativamente com POSIX. Para produtos Microsoft a API POSIX pode ser acessada através do Windows Services for UNIX.

A Tabela 2 apresenta as principais funções da API POSIX threads (Pthreads). Elas são largamente utilizadas por desenvolvedores C na criação de softwares multithreads.

2.9.2 *Threads* no RTMcarns

O sistema possui nove *threads*, uma para o Módulo de Cálculo, uma para o Módulo de Saída, três para o Gerenciador de Conexão, três para o Módulo de Entrada e a *thread*

Tabela 2: Funções da API POSIX *Threads*

Descrição	**POSIX *Thread* API**
Gerenciamento	Pthread_create,Pthread_exit,pthread_join, pthread_kill,pthread_self
Mutex	Pthread_mutex_init,Pthread_mutex_lock, Pthread_mutex_unlock,Pthread_mutex_destroy
Var. Condicionais	Pthread_cond_init,Pthread_cond_destroy, Pthread_cond_wait,Pthread_cond_signal,Pthread_cond_broadcast

principal que é iniciada com o processo do RTMcarns.

Ao iniciar o sistema quatros *threads* são imediatamente criadas: Módulo de Cálculo, Módulo de Saída, a principal do Módulo de entrada e a principal do Gerenciador de Conexão. Além de todos os semáforos mutex, variáveis condicionais de sincronismo e sinais.

Na Figura 12 pode-se observar um diagrama de atividade simplificado do sistema. O intuito desse diagrama é apenas mostrar a sequência de criação das *threads* e o sincronismo entre elas. Nesse diagrama foi omitida a *thread* de Gerenciador de Conexões, apenas por simplificação.

Após a criação das *threads* dos Módulos de Saída e de Entrada, elas entram imediatamente em estado de bloqueio, aguardando sinal do Gerenciador de Conexões. Esses sinais ocorrerão após algum cliente conectar-se às portas específicas do sistema.

O arquivo descritor da conexão é enviado aos módulos de interesse e então podem comunicar-se livremente com o cliente. O Módulo de Saída entra em estado de bloqueio novamente, porém aguardando o sinal do Módulo de Cálculo, que indica que os resultados já estão disponíveis para serem utilizados.

Após a Entrada estar conectada, cria mais duas *threads*: Leitura e Monitoramento. A *thread* de Leitura processa os dados que chegam pela conexão TCP, os valida e armazena em um buffer circular. O Monitoramento, como o nome diz, monitora o sistema de acordo com as configurações do usuário e parâmetros como intervalo entre os ciclos de cálculo ou a quantidade de dados que devem ser recebidos antes de iniciar um novo ciclo de cálculo. Após algum dos eventos monitorados ocorrerem, um sinal é enviado ao Módulo de Cálculo indicando que já existem dados suficientes para realização do cálculo.

Após a desconexão, as *threads* Entrada e/ou Saída voltam para o estado de bloqueio, esperando por uma nova conexão válida. Elas são independentes e uma pode estar em execução enquanto a outra está bloqueada esperando por conexão. Apesar de não ser tão interessante para a Saída executar se não existe dados de entrada, ela ficará bloqueada por falta de resultados. Entretanto, para a Entrada é indiferente o estado de conexão da Saída, ainda mais que os resultados gerados podem ser salvos em disco.

O Módulo de Cálculo espera o sinal vindo da Entrada, e então inicia o cálculo com

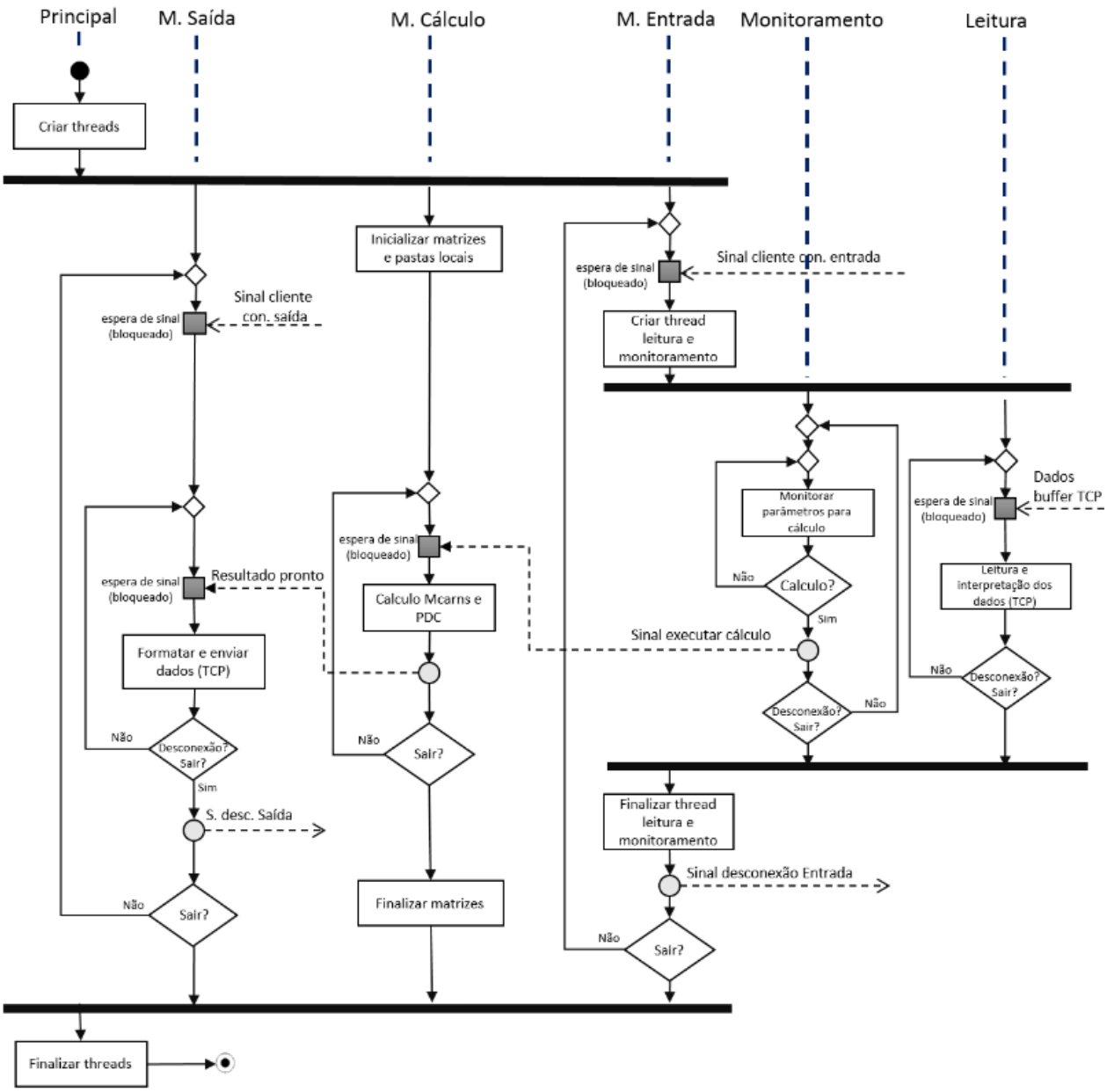

Figura 12: Diagrama de atividades simplificado do RTMcarns, acrescido de sinais de sincronismo e estado de bloqueio para as *threads*

os dados de entrada, após finalizá-los envia um sinal para a Saída indicando a existência de novos resultados.

Além da *thread* Gerenciador de Conexão, outras informações foram omitidas no diagrama por simplificação. O sistema ao ser encerrado, seja pelo usuário ou por alguma falha crítica, envia todos os sinais de sincronismo, a fim de liberar *threads* em bloqueio, após liberadas verificam se devem encerar a execução ou continuar com o trabalho.

Os sinais 'Sinal cliente con. saída' e 'Sinal cliente con. entrada' são enviados pelo Gerenciador de Conexões quando algum cliente TCP se conecta nas portas de saída e

entrada respectivamente.

2.10 Bibliotecas utilizadas

Como citado anteriormente, todo o sistema do RTMcarns foi desenvolvido em C e nenhuma biblioteca matemática adicional como BLAS, LAPACK ou SLICOT foi utilizada. Na Tabela 3 estão listadas todas as bibliotecas utilizadas no desenvolvimento do sistema, bem como suas dependências.

Tabela 3: Bibliotecas utilizadas no projeto

Dependências	**Bibliotecas**
Padrão C99	stdio.h, stdlib.h, string.h, math.h, complex.h, ctype.h, stdint.h
POSIX	pthread.h, unistd.h
Sistema Operacional Linux	fts.h, sys/socket.h, sys/types.h

Caso exista a necessidade da compilação para outra arquitetura de processador, mas ainda utilizando o Linux, nenhuma modificação será necessária. Porém se existir mudança no sistema operacional e o mesmo não for baseado em Unix, será necessário encontrar quais são as bibliotecas equivalentes as apresentadas na seção de "Bibliotecas do SO".

2.11 Gerenciador de Conexões

Este módulo monitora as portas TCP do sistema, identifica a conexão de um cliente e também presta a manutenção da reconexão caso exista uma desconexão. Logos após o seu início, outras duas threads são criadas: uma que configura e monitora a conexão para o Módulo de Entrada e outra que configura e monitora a conexão para o Módulo de Saída.

Como pode ser observado no diagrama de atividade da Figura 13, as *threads* Con. Entrada e Con. Saída criam um servidor TCP e entram no estado bloqueada, esperando alguma conexão nas porta configuradas. Após a conexão de um cliente TCP, o arquivo descritor (*file descriptor*) da conexão é copiado para uma variável compartilhada entre o Gerenciador de Conexão e os Módulos de Entrada e Saída. O nome dessa variável é 'config_con' e a estrutura dela pode ser observada no código abaixo:

```
1 typedef struct sock{
2         char on;
3         int socketnumber;
```

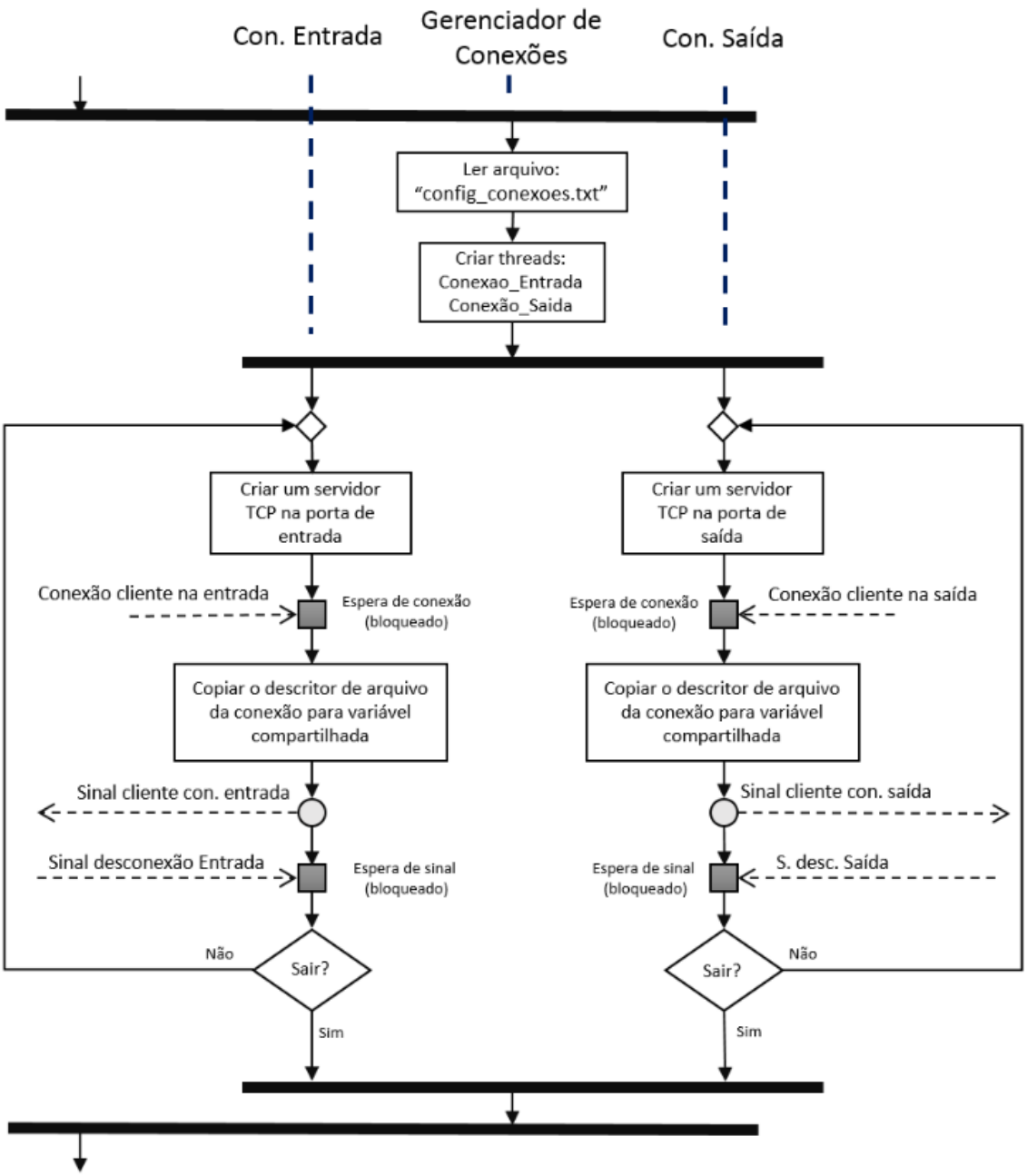

Figura 13: Diagrama de atividades simplificado do Gerenciador de Conexão

```
}sock_con;

typedef struct Config_Con{
        unsigned int port_Entrada;
        unsigned int port_Saida;
        unsigned int port_Config;
        unsigned int port_Debug;
        sock_con entrada;
        sock_con saida;
        sock_con config;
```

```
14          sock_con debug;
15 }TConfig_Con;
16
17 TConfig_Con config_con;
```

Esta estrutura foi criada para suportar quatro tipos de conexões, as de entrada e saída, e outras duas para implementação futuras: conexão para *debug* (em português depuração) e outra para alterar os parâmetros de configuração do sistema in *run-time*, ou seja, enquanto o RTMcarns estiver em execução.

Os campos port_Entrada, port_Saida, port_Config e port_Debug armazenam o número da porta TCP em que cada uma das conexões deve ser aberta. Os campos entrada, saída, config e *debug* são do tipo sock_con, e portanto possuem dois outros campos: um para indicar o estado da conexão e outro para armazenar o número do arquivo descritor da conexão.

Cada um dos campos do tipo sock_con são protegidos por semáforos do tipo mutex, eliminando problemas relacionados a concorrência pela variável, como acesso simultâneo.

```
1 pthread_mutex_t mutex_sockEntrada;
2 pthread_mutex_t mutex_sockSaida;
3 pthread_mutex_t mutex_sockConfig;
4 pthread_mutex_t mutex_sockDebug;
```

A implementação do Gerenciador de Conexões é simples e segura: após armazenar o *file descriptor* da conexão na variável compartilhada, um sinal é enviado para o Módulo correspondente da conexão. Dessa forma ele é desbloqueado e inicia a comunicação com o cliente através do descritor compartilhado.

Por sua vez a *thread* filha do Gerenciador de Conexões entra em estado bloqueada, e só retornará à execução após receber o sinal de conexão do Módulo correspondente. Caso exista uma desconexão, a *thread* filha será desbloqueada e iniciará um novo servidor TCP na mesma porta, que ficará aguardando a conexão de novos clientes.

2.12 Módulo de entrada

Como dito anteriormente, este módulo é responsável por receber os dados via conexão TCP estabelecida na porta de entrada e processá-los, além de enviá-los ao Módulo de Cálculo. Neste tópico apresenta-se detalhes de como ele funciona e sua interação com os outros Módulos. Para um melhor entendimento será abordado a interação com o Gerenciador, com o Módulo de Cálculo e os papéis das *threads* filhas: Leitura e Monitoramento.

2.12.1 Interação com o Gerenciador de Conexões

Como visto anteriormente, o Gerenciador e a Entrada comunicam-se através de dois sinais e uma variável compartilhada. Na Figura 14 pode-se observar em um único diagrama a relação entre os dois módulos. O ciclo sempre se repete:

a) Entrada e Gerenciador são criados;

b) Entrada fica bloqueada esperando sinal que indique uma conexão válida;

c) O Gerenciador cria as *threads* Con. Entrada e Con. Saída que manipulam as conexões;

d) Con. Entrada cria um servidor TCP e é bloqueado em um estado onde escuta a porta configurada para entrada;

e) Quando um cliente conecta na porta de entrada, o servidor armazena o *file descriptor* dessa conexão na variável 'config_con.entrada.socketnumber' e então envia o sinal para a Entrada;

f) O Módulo de Entrada é desbloqueado e então cria as *threads* Monitoramento e Leitura;

g) A Leitura e o Monitoramento ficam em execução até que o cliente desconecte da entrada;

h) As *threads* de Leitura e Monitoramento são finalizadas e o sinal indicando desconexão é enviado para a Con. Entrada;

i) O Módulo de Entrada volta a ser bloqueado esperando pelo sinal de conexão;

j) Con. Entrada é desbloqueada e cria novamente o servidor TCP na porta de entrada;

k) Repete-se o ciclo até que a execução do sistema seja finalizada.

2.12.2 Monitoramento

E a *thread* Monitoramento verifica, em intervalos fixos, o tempo decorrido do último ciclo de cálculo e a quantidade de dados novos que foram inseridos na matriz de leitura. De acordo com as configurações do usuário, ela inicia o processo de "envio" da matriz de entrada para o Módulo de Cálculo. Um diagrama mais detalhado da *thread* de Monitoramento pode ser observado na Figura 15.

Enquanto a *thread* Leitura realiza o *parser* (interpretação) dos dados que chegam pela TCP e insere os mesmo na matriz leitura, a *thread* Monitoramento apenas espera até o momento de enviar o sinal de execução de cálculo para o Módulo de Cálculo. Entretanto, antes de enviar esse sinal, o Monitoramento realiza algumas operações.

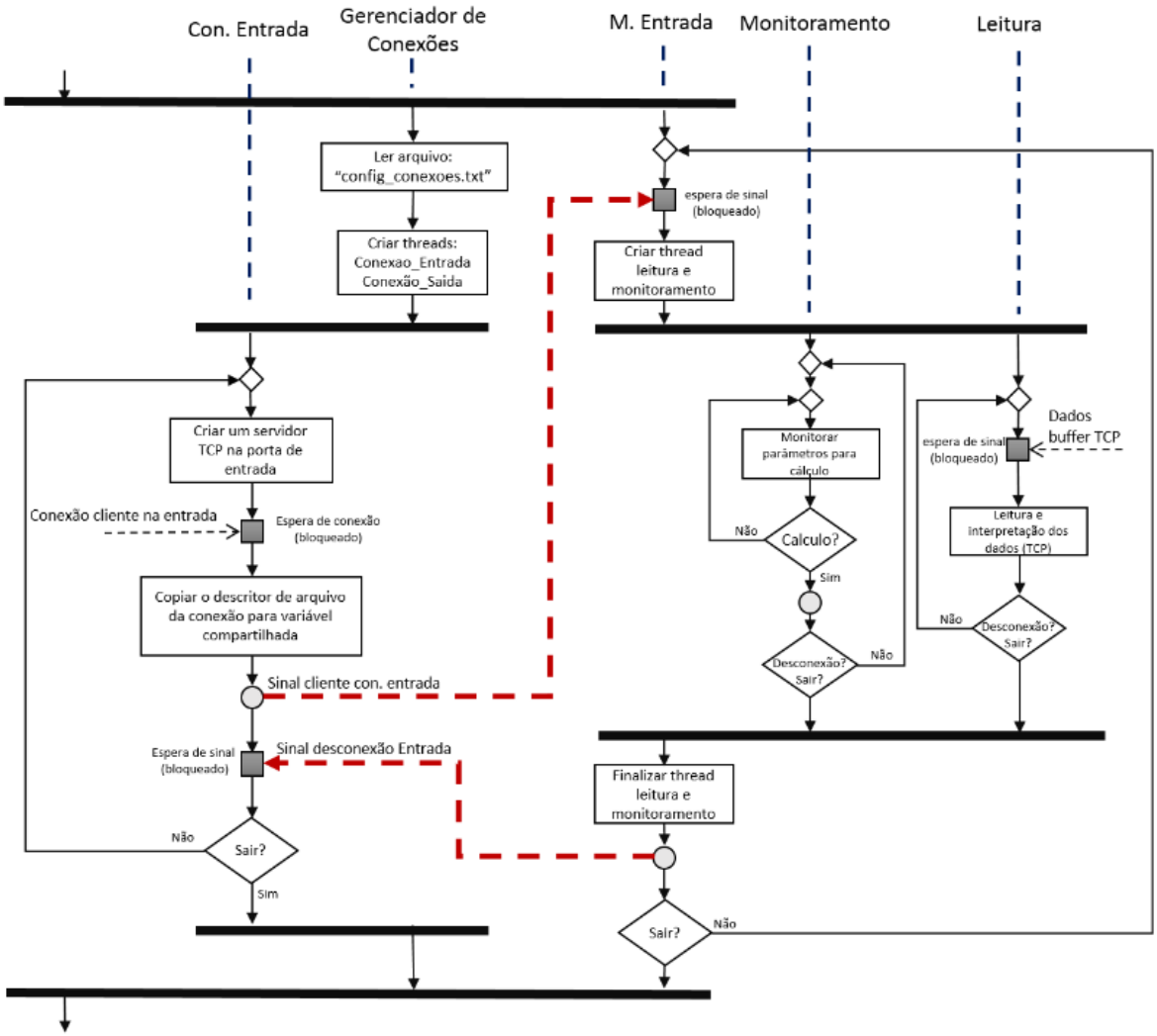

Figura 14: Diagrama de atividades simplificado da interação entre o Módulo de Entrada e o Gerenciador de Conexões

O primeiro passo é obter acesso a matriz leitura, que por ser um recurso compartilhado com a *thread* Leitura, deve obter o mutex 'mutex_leitura'. Caso não seja possível, a *thread* entrará em bloqueio aguardando a liberação do mutex. Quando o Monitoramento conseguir acesso a matriz leitura, ele a copia para uma variável temporária a fim de liberá-la rapidamente, para que não afete negativamente o comportamento da *thread* Leitura.

Essa cópia da matriz são os dados que serão utilizados no ciclo atual de cálculo. Dependendo da configuração do usuário, ela irá compor uma matriz maior que possui dados de ciclos passados.

Pode existir uma diferença entre o tamanho da matriz que deve ser utilizada para o cálculo e a quantidade de amostras novas, isso depende da configuração do usuário no arquivo 'config_Algoritmo'. Neste caso significa que o usuário quer que seja utilizado dados de ciclos anteriores no cálculo, sendo assim a matriz entrada é montada com uma junção

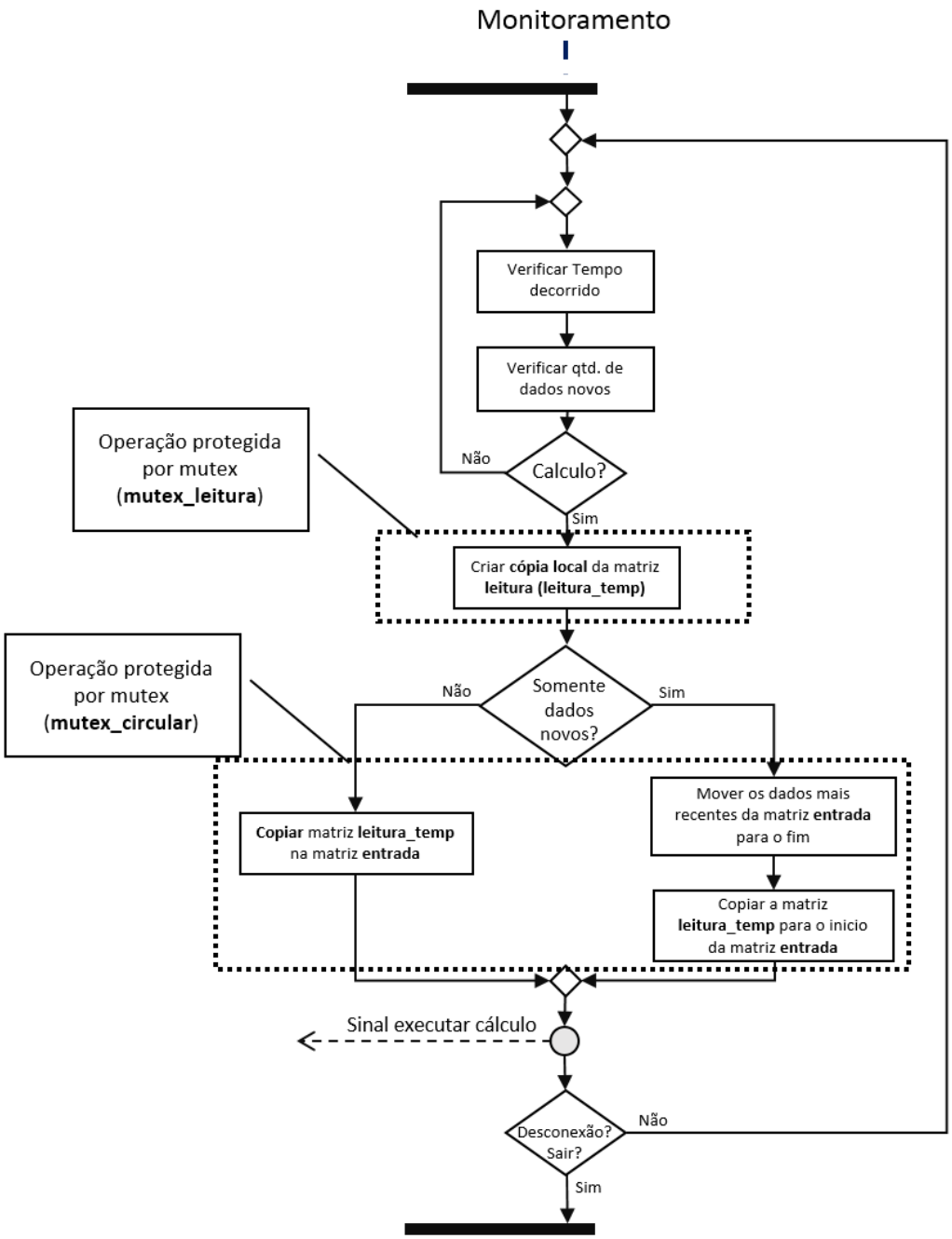

Figura 15: Diagrama de atividades da *thread* Monitoramento

da cópia da matriz leitura e dos dados anteriores da própria matriz entrada, exemplificado na Figura 16 está o comportamento com um vetor de dados, ou seja, apenas um canal, o mesmo ocorre para todos os vetores da matriz.

2.12.3 Leitura

Basicamente, a *thread* de Leitura recebe os dados do *buffer* (armazenamento temporário de dados) TCP e os armazena em na matriz leitura que possui a estrutura de

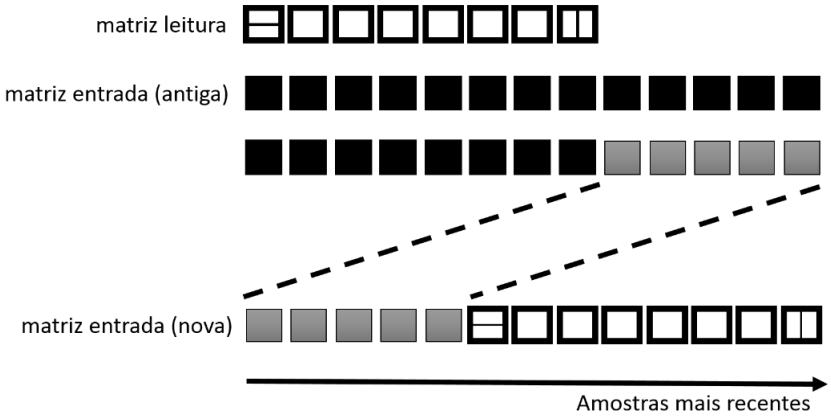

Figura 16: Junção da matriz leitura com a matriz entrada

controle de um *buffer* circular.

A matriz leitura foi projetada como um *buffer* circular, pois algumas configurações do usuário podem gerar situações em que o intervalo entre os ciclos de cálculo seja muito extenso comparado ao fluxo de dados na entrada. E, se o usuário configurou sua preferência pelo intervalo de tempo e não quantidade de amostras, então os dados que chegarem pela porta TCP começarão a sobrescrever os já existentes na matriz de leitura. A Figura 17 ilustra esse comportamento, em que os dados que chegam na TCP sobrescrevem os já existentes na matriz leitura, devido ao controle circular.

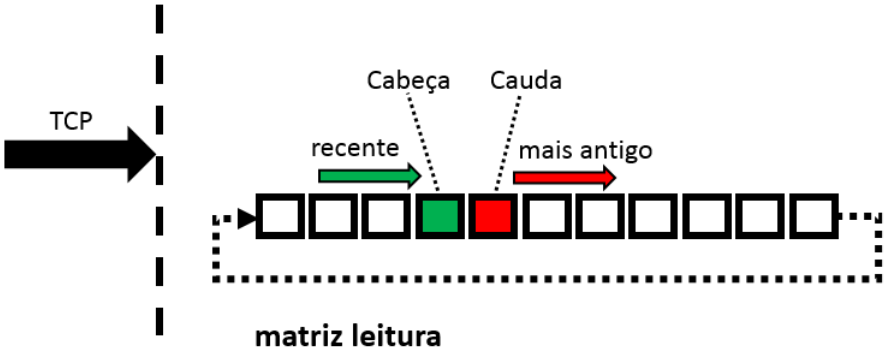

Figura 17: Matriz leitura com controle circular

Em um cenário que o fluxo de sinal na entrada é maior que o consumo pelo Módulo de Cálculo, dados serão perdidos de uma forma ou outra, restando assim a escolha entre: ignorar os dados que chegam pela TCP após preencher a matriz leitura com novos dados, ou optar por sempre manter na matriz os dados mais atuais vindo da TCP. Esse trabalho optou, por enviar para o cálculo os dados mais recentes do usuário que chegaram pela TCP, implementando o comportamento de *buffer* circular na matriz leitura.

Na Figura 18, pode-se observar o diagrama de atividade da *thread* Leitura, bem

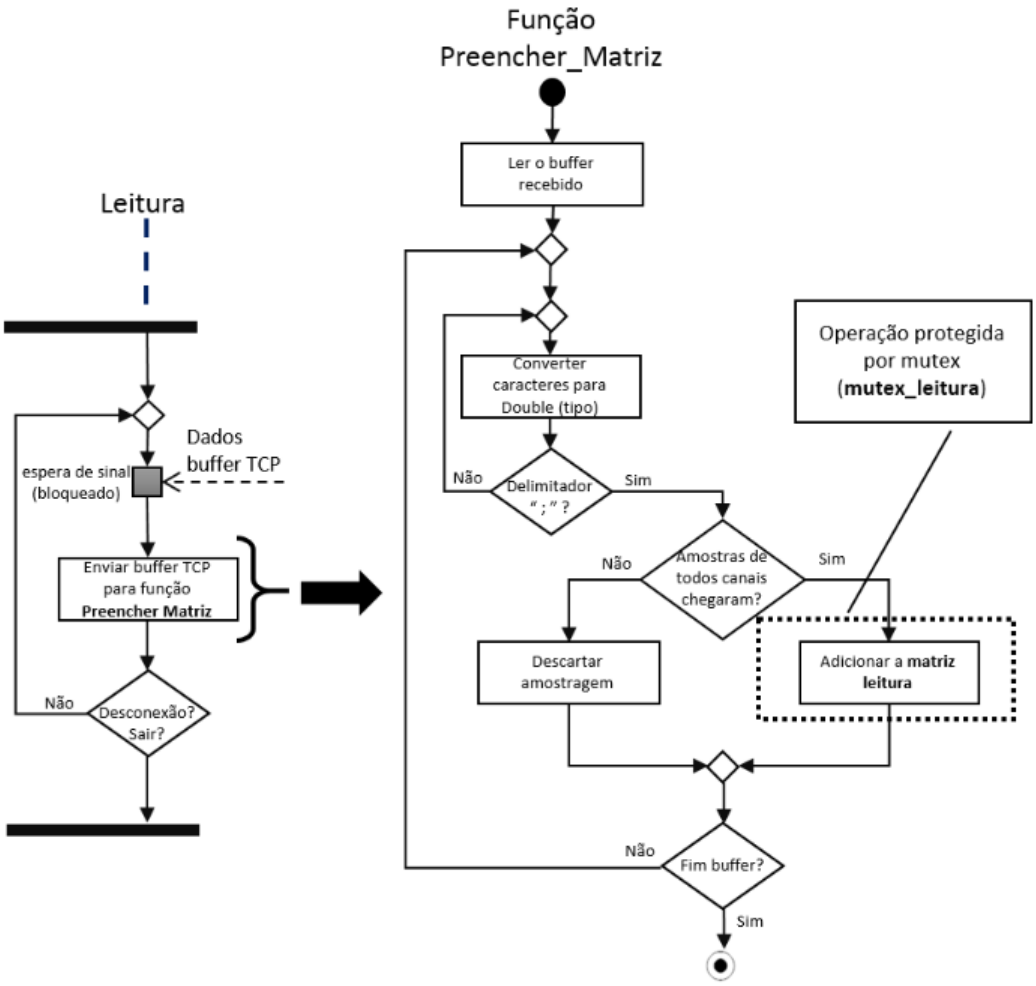

Figura 18: Diagrama de atividade da *thread* Leitura e da função Preencher_Matriz

como da função Preencher_Matriz, que é responsável por interpretar os caracteres que chegam pela TCP e transforma-los em valores numéricos do tipo *Double* (ponto flutuante de dupla precisão), além de verificar se as amostras de todos os canais foram enviadas. Caso a amostragem esteja completa, os valores são incluídos na matriz leitura que, como dito anteriormente, é compartilhada com a *thread* Monitoramento.

Atualmente, os dados enviados para o RTMcarns devem possuir o formato mostrado em tópicos anteriores, que são puramente caracteres trafegando via TCP. Como parte da modularidade do sistema, apenas essa função precisa ser reescrita para que novos formatos de dados sejam aceitos e interpretados pelo sistema. Basta manter a interface com o sistema, que é a entrada de um *buffer* e preencher a matriz leitura com os dados interpretados.

2.13 Módulo de Cálculo

O Módulo de Cálculo é separado em dois grandes algoritmos: Mcarns e PDC.

O Algoritmo Mcarns estima o modelo autorregressivo multivariado de um sinal multicanal utilizando o método de Nuttall-Strand. Essas estimações são a entrada para o segundo passo do cálculo, o PDC, que calcula a matriz de coerência parcial direcionada, um qualificador de conectividade.

Sendo o RTMcarns um sistema que visa fornecer uma plataforma de estudo para algoritmos de conectividade cerebral em tempo real, este é o módulo que oferece as maiores oportunidades de estudos futuros, pois novos algoritmos podem ser implementados sem nenhuma mudança no funcionamento do sistema, seja um novo algoritmo para estimar o modelo de regressão linear, outro método para calcular qualificadores de conectividade ou qualquer outra função utilizada nos cálculos.

Antes de entrar em detalhes sobre implementação dos algoritmos, será discutido o funcionamento geral deste módulo e como ele interage com os outros módulos.

Na Figura 19 observa-se o diagrama de atividades mais detalhado do Módulo de Cálculo. Após o sistema ser inicializado, ele cria as pastas locais. Caso as pastas já existam, serão apagadas, ou seja, o RTMcarns sempre sobrescreverá os arquivos locais da execução anterior.

O Módulo de Cálculo precisa esperar a matriz entrada ficar pronta para que possa iniciar a execução dos algoritmos, então ficará bloqueado aguardando o sinal que será enviado pela *thread* de Monitoramento quando for o momento de iniciar o ciclo de cálculo.

Com a chegada do sinal o módulo é desbloqueado e realiza uma cópia local da matriz entrada, para que seja passada para a função Mcarns estimar o modelo de regressão multivariado da dos dados da matriz entrada. As matrizes de coeficientes autorregressivos resultante da estimação do Mcarns são enviadas para a função que calcula o PDC e a matriz resultante é uma variável que é compartilhada com o Módulo de Saída.

Assim, para que o Módulo de Cálculo possa gravar localmente as matrizes de resultado em csv, sem que isso atrapalhe o comportamento do Módulo de Saída, realiza-se uma cópia dessas matrizes antes de enviar o sinal 'Resultado Pronto'. Lembrando que nesse projeto qualquer variável compartilhada é protegida por um mutex.

Após enviar o sinal para o Módulo de Saída, as matrizes serão salvas no disco local, caso o usuário tenha configurado para tanto. Se o sistema não estiver sendo encerrado o Módulo de Cálculo entrará em estado bloqueado novamente, aguardando o próximo sinal da *thread* de Monitoramento para iniciar um ciclo de cálculo.

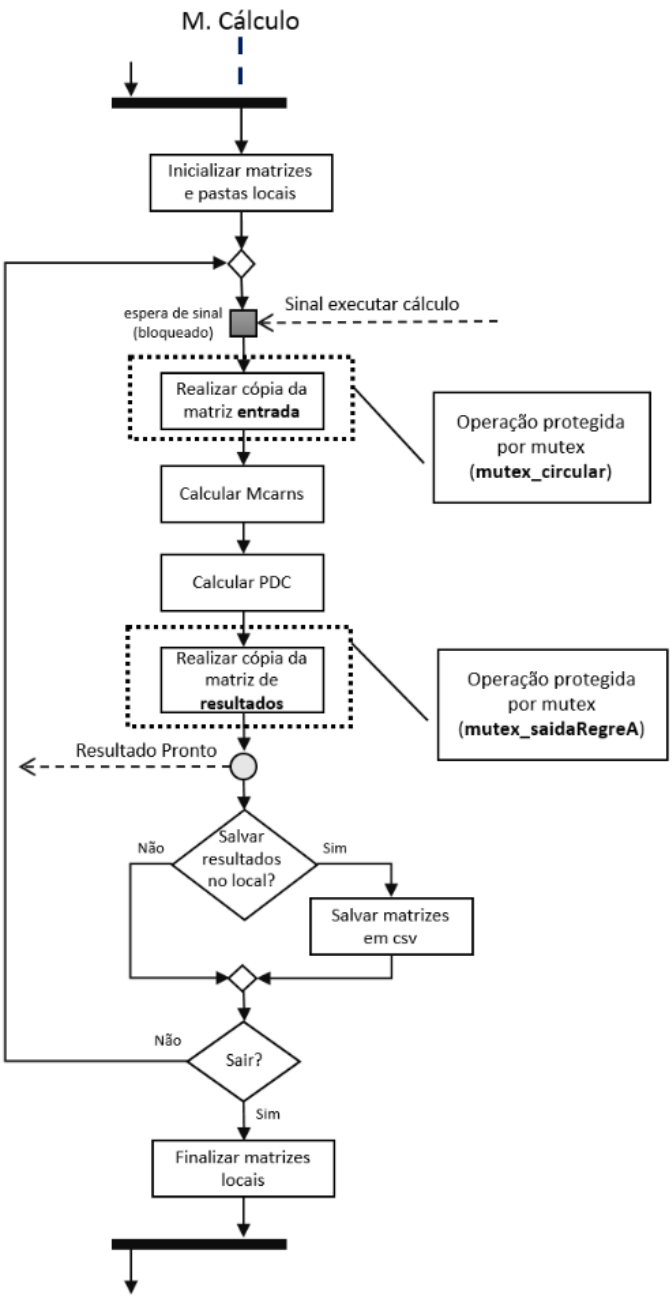

Figura 19: Diagrama simplificado de atividade do Módulo de Cálculo

2.13.1 Detalhando o Módulo de Cálculo

Todos as subrotinas utilizadas pelas funções Mcarns e PDC foram desenvolvidas em C e muitas delas podem não estar implementadas da melhor forma possível e nem utilizando todos os recursos que o hardware fornece. A simples substituição de uma delas pode ter um impacto considerável no desempenho do sistema, e esse tipo de experiência na plataforma deve ser incentivado para que um dia o RTMcarns chegue a um "estado de arte" em sua implementação. Neste tópico será abordado alguns detalhes pertinentes da implementação do Módulo de Cálculo.

2.13.1.1 Estrutura de arquivos

É importante entender como a implementação do Módulo de Cálculo está distribuído entre os arquivos C e *headers* (arquivo de cabeçalho do C). Observe na Figura 20, os principais arquivos que compõe este módulo.

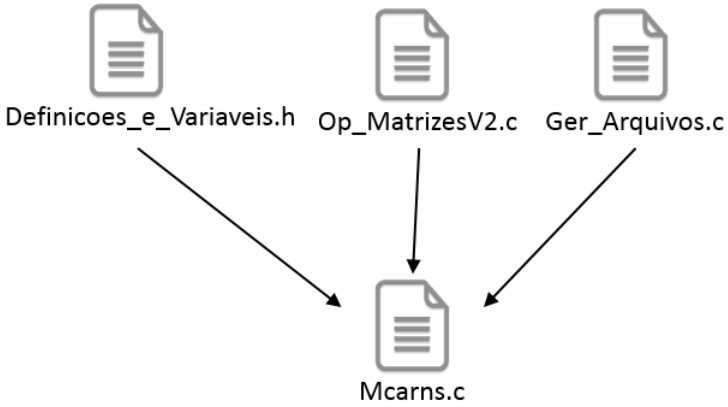

Figura 20: Principais arquivos para o Módulo de Cálculo

No arquivo 'Mcarns.c' estão implementados os algoritmos Mcarns e PDC, no *header* 'Definicoes_e_Variaveis.h' estão definidos todos os tipos de estruturas e variáveis para o sistema, no 'Op_MatrizV2.c' estão todas as implementações de funções algébricas e no arquivo 'Ger_Arquivos.c' as funções que manipulam arquivos, tanto leitura como escrita.

Não é recomendado modificar parâmetros no arquivo 'Definições_e_Variaveis.h' sem que exista um profundo conhecimento da implementação do sistema, dado que esse arquivo pode modificar o correto funcionamento de todo o sistema.

Quanto ao 'Ger_Arquivos.c', deve ser alterado caso deseje adicionar novas extensões de arquivos para salvar as matrizes resultantes dos cálculos. Novas funcionalidade devem ser adicionadas, evitando modificar as existentes para não causar o mal funcionamento de partes do sistema que dependem da manipulação de arquivos.

A maior parte das alterações devem ficar para o "Op_MatrizV2.c", pois é a que contém todas as funções algébricas para a manipulação das matrizes, e possui um impacto direto na precisão dos resultados e na performance do sistema.

2.13.1.2 Tipo dos dados

A primeira tipo apresentado é o "matrizV2", pois é a estrutura base para outros tipos e é o parâmetro de entrada e saída da maioria das funções. A sua declaração pode ser vista no trecho de código a seguir.

```
1 struct Matriz_Dinamica {
```

```
2     int lin;
3     int col;
4     double **mat;
5 };
6 typedef struct Matriz_Dinamica matrizV2;
```

Basicamente, essa estrutura cria uma variável que possui lin e col, inteiros utilizados para armazenar o número de linhas e colunas, respectivamente. O campo mat é onde será armazenado todos os valores da matriz, ela é um ponteiro para outro ponteiro e sua locação de espaço na memória é dinâmico, evitando desperdício de memória.

Ponteiro é uma variável que armazena um endereço de memória e como todo objeto na memória possui um endereço, um ponteiro pode armazenar esse endereço e através dele acessar os valores da variável para o qual ele aponta.

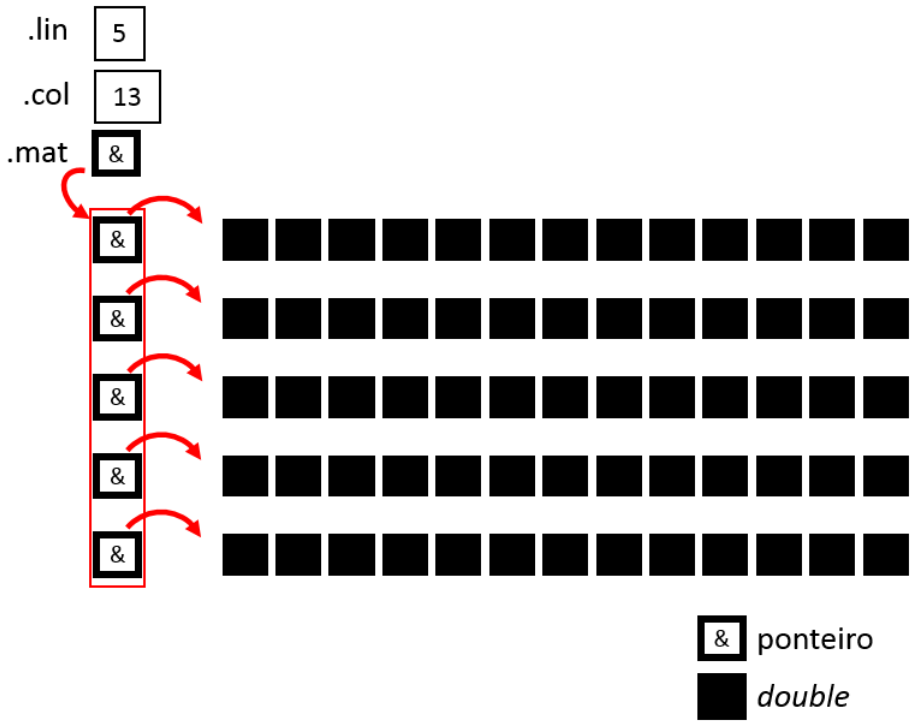

Figura 21: Ilustração de uma variável do tipo matrizV2

O fato é que mat aponta para um vetor de ponteiros, que por sua vez aponta para um vetor de *double*, assim como mostrado na Figura 21.

Para a criar e destruir essas matrizes foram criadas funções, a fim de facilitar a interação com esse tipo de variável e evitar vazamento de memória.

```
1 void MLoc_Matriz(int lin, int col, double **p);
2
3 void Free_Matriz(int lin, int col, double **p);
```

A função 'MLoc_Matriz' aloca memória dinamicamente os vetores de *double* de cada ponteiro. Porém, para utilizá-la, o vetor de ponteiros deve ter sido inicializado com o número de linhas da matriz.

O trecho de código a seguir exemplifica a criação, manipulação e destruição de uma variável do tipo 'matrizV2'.

```
1 /* Declare a matriz */
2 matrizV2 A;
3
4 /* Exemplo: 2 canais com 3 amostras cada */
5 /* Crie a matriz */
6 A.lin=2;
7 A.col=3;
8 A.mat = ( double** )( malloc( A.lin * sizeof(double*) ) );
9 MLoc_Matriz( A.lin , A.col , A.mat);
10
11 /* Utilize a matriz */
12 A.mat[0][0] = 1;
13 A.mat[1][2] = 12;
14
15 /* Destrua a matriz */
16 Free_Matriz( A.lin , A.col , A.mat);
```

As funções implementadas no 'Op_MatrizesV2.c' recebem um ponteiro para uma 'matrizV2' como entrada e o também como saída. Para exemplificar seguem as declarações de algumas funções do 'Op_MatrizesV2'.

```
1                                   ...
2 void Matriz_Inversa_V2( matrizV2 *M , matrizV2 *M_out );
3 void Matriz_Cofatores_V2 (matrizV2 *M , matrizV2 *M_out);
4 void Menor_Complementar_V2(matrizV2 *M,int c_lin,int c_col,matrizV2 *M_out);
5 void Matriz_Adjunta_V2( matrizV2 *M , matrizV2 *M_out);
6 void Transposta_V2( matrizV2 *M , matrizV2 *M_out);
7 void Kron_lyap_V2( matrizV2 *A, matrizV2 *B, matrizV2 *C, matrizV2 *M_out );
8                                   ...
```

A função Mcarns recebe como parâmetro de entrada uma variável do tipo 'matrizV2' que possui os dados de entrada, e também um inteiro com o valor da ordem do modelo autorregressivo multivariado que deve ser calculado. O resultado dessa função é um conjunto de matrizes que possuem os coeficientes do modelo dessa regressão, foi criado um tipo para armazenar essas matrizes chamado "matrizRegressores"que baseia-se em variáveis do tipo 'matrizV2'. Veja no trecho de código a seguir.

```
1 #define LIM_Regressores 128
2 struct Matriz_Resultados {
3         matrizV2 pf;
4         matrizV2 pb;
5         matrizV2 ef;
6         matrizV2 eb;
7         matrizV2 A[LIM_Regressores];
8         matrizV2 B[LIM_Regressores];
9 };
10 typedef struct Matriz_Resultados matrizRegressores;
```

Observe que dentro da estrutura 'matrizRegressores' existem sete variáveis, pf, pb, ef e eb são matrizes bidimensionais do tipo 'matrizV2', e, A e B que são vetores que possuem 128 matrizes do tipo 'matrizV2'. A quantidade de matrizes de A e B dependem da ordem do modelo autorregressivo escolhida pelo usuário e o máximo é 128 definido em 'LIM_Regressores'. Caso seja necessário ordens superiores basta aumentar esse valor.

E a última grande estrutura é a 'matrizFinal', que armazena os valores de coerência parcial direcionada resultantes da execução do algoritmo de PDC. Ela é uma matriz tridimensional, como mostrado no trecho de código a seguir.

```
1 struct Matriz_Final {
2         int lin;
3         int col;
4         double ***mat_d;
5 };
```

Também foram implementadas funções para a criação e destruição dessa matriz. Abaixo um exemplo de código.

```
1 /* Declare a matriz */
2 matrizFinal R_Final;
3
4 /* Crie a matriz */
5 R_Final.col = 5; //n. de canais
6 R_Final.lin = 5; //n. de canais
7 R_Final.mat_d = ( double*** )malloc( R_Final.lin*sizeof(double**) );
8 n_fft = 128;
9 MLoc_Matriz_final( R_Final.lin , R_Final.col , n_fft , R_Final.mat_d);
10
11 /* Utilize a matriz */
12 R_Final.mat_d[0][1][1] = 10;
13
14 /* Destrua a matriz */
15 Free_Matriz_final(R_Final.lin , R_Final.col , R_Final.mat_d);
```

Essa matriz final possui as dimensões: (número de canais)x(número de canais)x quantidade de raias utilizadas no cálculo da FFT (*Fast Fourier Transform*) no passo do PDC.

2.13.1.3 Inversão de matriz

A operação de inversão de matriz é uma operação custosa e sua implementação tem influência direta no desempenho total do sistema, por ser utilizada muitas vezes a cada ciclo de cálculo durante a estimação do modelo de regressão.

Neste trabalho optou-se por um algoritmo de inversão por blocos, técnica criada por Han Boltz (MATHEW et al., 2012) em 1923 e generalizada por Tadeusz Banachiewicz (TADEUSZ, 1937) em 1937 , quem também provou sua validade.

Essa é uma técnica para inversão de grandes matrizes de maneira eficiente. A inversão por blocos utiliza a equação (2.1) para inverter uma matriz:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix} \tag{2.1}$$

onde:
A e D devem ser quadradas, e a dimensão de A deve ser igual ou maior que D.

Neste método precisamos calcular apenas a inversa de A, que é chamada de grande bloco, e de $(D - CA^{-1}B)$, que é a matriz Schur complementar de A (COTTLE, 1974).

O método de inversão por bloco é naturalmente recursivo e sem maiores dificuldades pode ser implementado com a técnica de divisão e conquista para solucionar a equação.

Divisão e conquista é uma técnica de projeto de algoritmos utilizada pela primeira vez por Anatolii Karatsuba. Essa técnica consiste em quebrar o problema recursivamente em problemas menores até que esteja em uma dimensão possível de se resolver diretamente. E, com a combinação dos resultados obtidos da solução dos problemas menores, é possível chegar na solução do problema inicial (CORMEN, 2009).

Utilizando o método de inversão em blocos com a técnica de divisão e conquista, basta dividir a matriz em blocos, recursivamente, até que chegue em uma dimensão que seja simples calcular a inversa. Neste trabalho todas as matrizes são divididas até que o bloco A tenha dimensão menor ou igual a 3, para que então seja invertida utilizando a

solução analítica como mostrado na equação (2.2).

$$A^{-1} = \frac{1}{|A|}C^T = \frac{1}{|A|}\begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m,1} & C_{m,2} & \cdots & C_{m,n} \end{pmatrix} \tag{2.2}$$

onde:
|A| é o determinante de A, C é a matriz de cofatores e C^T a transposta.

Na Figura 22 se vê um exemplo da árvore formada pela técnica de divisão e conquista para inverter uma matriz 20x20, utilizando o método de inversão por blocos.

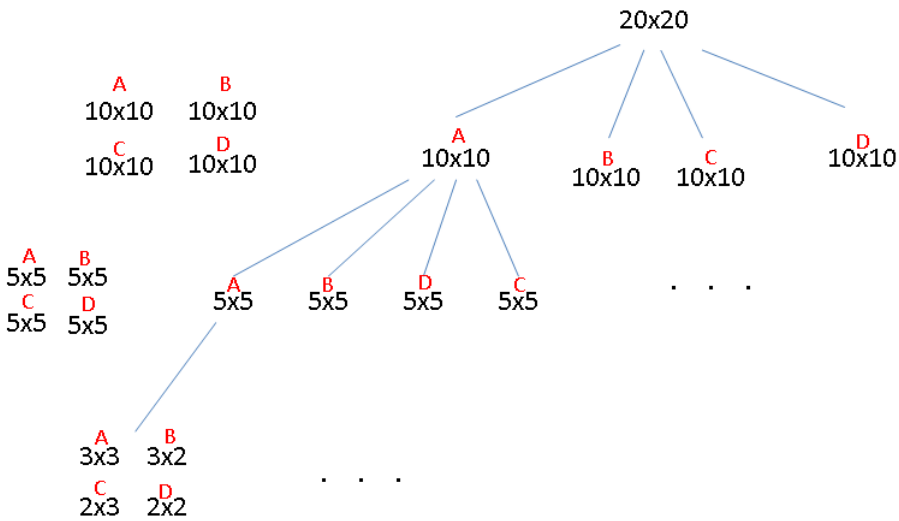

Figura 22: Inversão por blocos

Ao chegar na folha da árvore utiliza-se a solução analítica para calcular A^{-1} e $(D - CA^{-1}B)^{-1}$, e a solução dos blocos maiores será através de combinações dos resultados dos menores seguindo a equação (2.1).

Essa técnica pode tirar proveito de processadores com múltiplos núcleos, pois a fase de divisão do algoritmo cria problemas menores e isso proporciona uma distribuição natural do trabalho. Esses trabalhos menores podem ser distribuídos pelos processadores disponíveis. Neste trabalho esse recurso não foi implementado, mas espera-se que em trabalhos futuros crie-se implementações paralelas dessa técnica, a fim de utilizar todo o potencial da solução.

Desde de que a inversão por bloco de uma matriz nxn requer 2 inversões de metade do tamanho $(\frac{n}{2})$ e 6 multiplicações entre matrizes de dimensão $\frac{n}{2}$, pode ser mostrado que utilizar o método de inversão por bloco com a técnica de divisão e conquista, implica

em uma complexidade computacional igual ao algoritmo de multiplicação de matrizes utilizado internamente (CORMEN, 2009). Neste trabalho não foi implementado nenhuma técnica específica para otimizar o produto entre matrizes, tendo então apenas o algoritmo padrão que possui complexidade O(n^3).

Existem outros algoritmos para multiplicação de matrizes que possuem complexidade menor e podem ser implementados para obter melhor desempenho do sistema, como Strassen *algorithm* com O($n^{2,807}$) e Coppersmith-Winograd *algorithm* com O($n^{2,376}$) (DAVIE; STOTHERS, 2013).

Há também a possibilidade de, alterar o método de inversão e implementar outro algoritmo para comparar o desempenho do sistema.

2.13.1.3.1 O problema da recursividade

Neste trabalho foram consideradas as implicações de se ter uma função recursiva implementada no sistema, e é de comum conhecimento que esse tipo de solução deve ser evitado ao máximo, mas a possibilidade de trabalhar com um método paralelizável somado ao *hardware* disponível para implementação, foram fatores determinantes para arriscar em uma solução inicial recursiva para inversão de matriz.

Stack é uma região da memória para armazenamento de dados temporários, como retornos de funções e variáveis locais.

O problema de uma implementação recursiva é que, dependendo da função e da quantidade de chamadas realizadas, o *stack* pode ser consumido rapidamente.

As arquiteturas de processadores atuais armazenam o *stack* em RAM, e o tamanho do mesmo varia conforme a necessidade. A solução atual com a placa IMX6 possui 2GB de RAM, e isto foi um fator atenuador para o problema da recursividade. Caso exista a possibilidade de portar a solução para outras plataformas, esse item deve ser analisado com cuidado, pois caso a RAM seja limitada ou o *stack* implementado em *hardware*, então outra solução para inversão de matrizes deve ser estudada.

Apenas como referência, pode-se estimar quantas chamadas a função terá em um mesmo *stack* pela dimensão da matriz, ou seja, quantas chamadas são necessárias para que um dos ramos da árvore chegue até a folha, sendo que a recursão da inversão por bloco termina quando o bloco A for uma 3x3 ou menor, então foi montado uma tabela a partir da dimensão 3 e multiplicando por 2 a cada passo.

Para dimensões não existentes na Tabela 4, deve-se considerar sempre o valor acima. Para uma matriz 250x250, por exemplo, que está entre 7 e 8 chamadas até a folha, deve-se considerar o maior, sendo então 8 chamadas necessárias em um mesmo *stack* para a solução o menor problema da árvores, segundo então com a fase de combinações.

Tabela 4: Número de chamadas necessárias para solução de uma matriz de tamanho nxn

Qtd. de chamadas	**Dimensão da matriz** (n)
1	3
2	6
3	12
4	24
5	48
6	96
7	192
8	384
9	768
10	1536
11	3072
12	6144

Veja na Figura 23 o gráfico criado a partir dos dados da tabela. E ele foi gerado com o auxílio de um *software* matemático e a equação $y = 1,4427ln(n) - 0,585$ pode ser utilizada para extrapolar a Tabela 4.

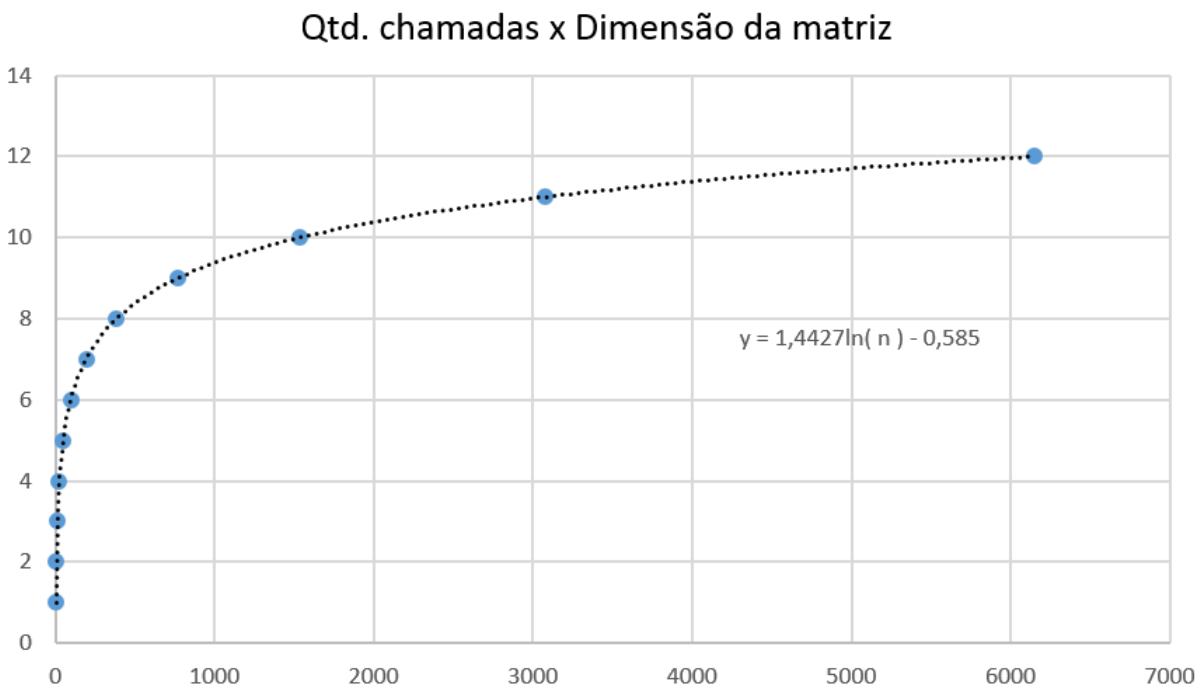

Figura 23: Estimativa de chamadas de função pela dimensão da matriz a ser invertida

2.14 Módulo de Saída

Assim como o Módulo de Entrada, a Saída comunica-se com o Gerenciador de Conexões através de sinais de sincronismo, para saber quando existe uma conexão ativa

na porta TCP de saída. Veja na Figura 24, como é a interação entre o Módulo de Saída e o Gerenciador de Conexões.

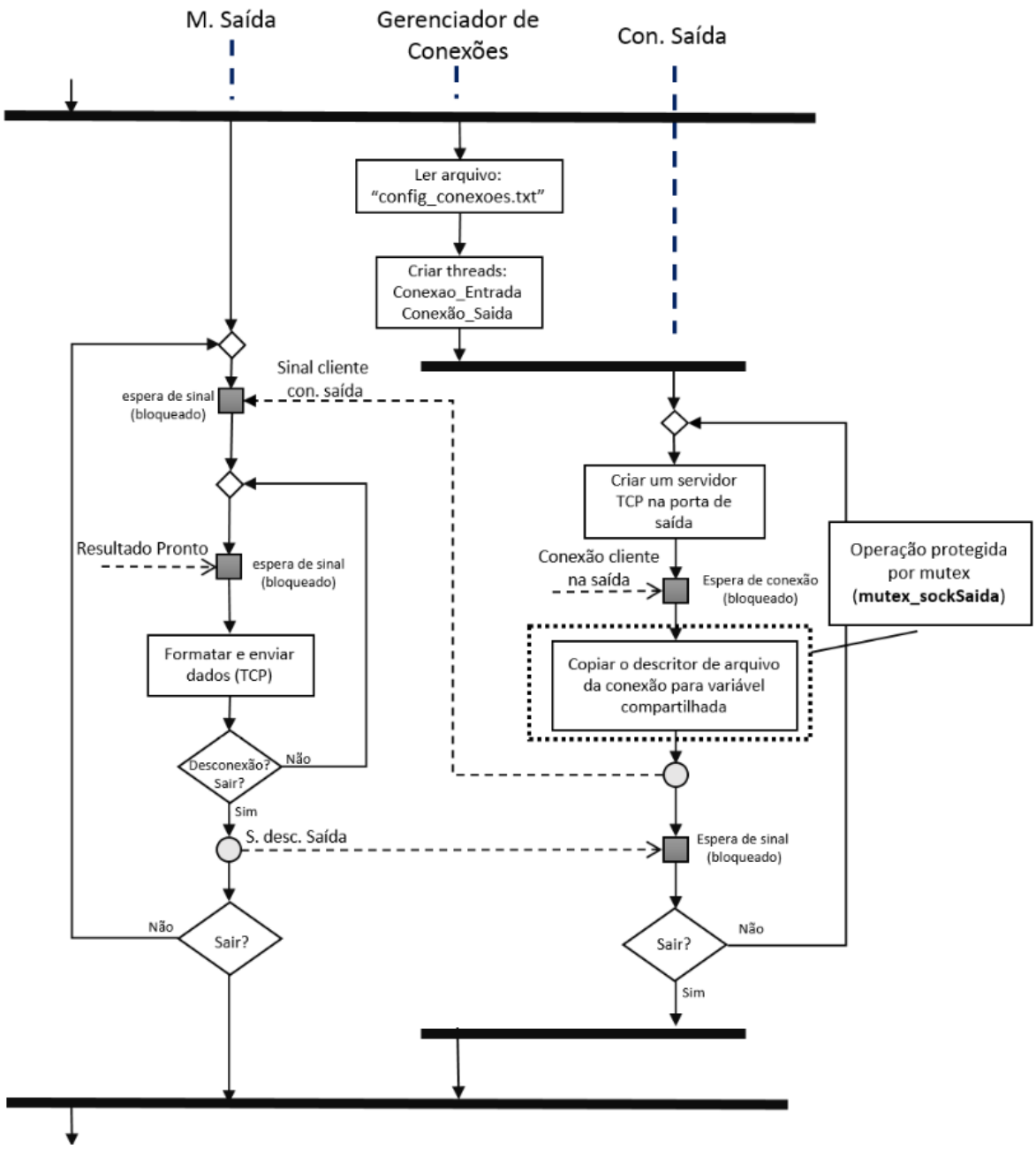

Figura 24: Diagrama de atividade da interação entre o o Módulo de Saída e o Gerenciador de Conexões

Analogamente ao Módulo de Entrada, o ciclo de manutenção da conexão do Módulo de Saída segue os seguintes passos:

a) Saída e Gerenciador são criados;

b) Saída fica bloqueada esperando sinal que indique uma conexão válida;

c) O Gerenciador cria as *threads* Con. Entrada e Con. Saída, que manipulam as conexões;

d) Con. Saída cria um servidor TCP e é bloqueado em um estado em que escuta a porta configurada para saída;

e) Quando um cliente conecta na porta de saída, o servidor armazena o *file descriptor* dessa conexão na variável 'config_con.saida.socketnumber' e então envia o sinal para a Saída;

f) O Módulo de Saída é desbloqueado pela conexão, porém é bloqueado novamente esperando o sinal de sincronismo do Módulo de Cálculo;

g) Sempre que existir resultados novos gerados pelo Módulo de Cálculo, a Saída recebe o sinal, formata o resultado e envia pela TCP;

h) Caso exista desconexão na porta TCP da saída, um sinal é enviado para o Gerenciador de Conexões indicando a desconexão;

i) O Módulo de Saída volta a ser bloqueado esperando pelo sinal de conexão;

j) Con. Saída é desbloqueada e cria novamente o servidor TCP na porta de saída;

k) Repete-se o ciclo até que a execução do sistema seja finalizada.

Este é o módulo mais simples do RTMcarns. E ele basicamente aguarda o sinal de sincronismo que é enviado pelo Módulo de Cálculo para iniciar a transmissão dos resultados via TCP. Veja a seguir a principal função deste módulo.

```
1 int EnviarMatrizSocket(matrizV2 *A , int sockfd , char cabecalho , int index)
```

A função 'EnviarMatrizSocket' é responsável por enviar qualquer matriz do sistema por uma conexão TCP válida, lembrando como a saída é formatada:

$$< \textbf{ID} < \textbf{índice} < v_{11}\ v_{12}\ v_{13}\ v_{14}\ ;\ v_{21}\ v_{22}\ v_{23}\ v_{24}\ ;\ v_{31}\ v_{32}\ v_{33}\ v_{34}\ ;\ v_{41}\ v_{42}\ v_{43}\ v_{44} >$$

onde, **ID** $\in [A, W, B, X]$, **índice** $\in [0 : (p-1)]$ e os valores enviados são de uma matriz 4x4.

Então, o parâmetro **A** é uma matriz qualquer do tipo 'matrizV2', o **socktfd** é o *file descriptor* da conexão TCP, **cabecalho** é o ID e **index** é o indice da cadeia de caracteres que será enviada.

Para alterar o formato da saída do RTMcarns, basta modificar a função 'EnviarMatrizSocket' e todas as matrizes serão enviadas no novo formato para o dispositivo conectado na saída.

Parte IV

Testes e resultados

3 Testes e resultados

Este tópico abordará como as funções criadas para o RTMcarns foram validadas quanto a precisão de seus resultados. Também apresentará um comparativo de desempenho entre a implementação do algoritmo no MatLab e do RTMcarns, além de comparativos entre diferentes implementações do RTMcarns.

3.1 Precisão numérica do sistema

3.1.1 Funções para importar e exportar csv

Estas funções são apresentadas no capítulo 3, pois foram criadas especificamente para testar a implementação do RTMcarns com um conjunto de dados externos, importados de um software matemático que foi utilizado de referência por todo o desenvolvimento do projeto.

As sub-rotinas criadas para esse propósito foram:

```
1 void Ler_Matriz_From_CSV_V2(matrizV2 *dados, const char *nome_arq);
2 void Criar_CSV_Apartir_Matriz_V2(matrizV2 *dados, const char *nome_arq);
3 void Criar_CSV_Vetores_RFinal_3DV2(matrizFinal *dados,const char *nome_arq);
```

A função 'Ler_Matriz_From_CSV_V2', é utilizada para importar matrizes a partir de um arquivo csv, preenchendo uma variável do tipo 'matrizV2' com os dados lidos do arquivo.

Basicamente, o primeiro parâmetro é um ponteiro para a variável matriz que deverá ser preenchida e o segundo é o caminho até o arquivo (*file path*).

O método 'Criar_CSV_Apartir_Matriz_V2' exporta uma variável 'matrizV2' para um csv. Ele deve receber o ponteiro para a matriz que se deseja exportar e o caminho completo com o nome que o arquivo deve ser salvo.

Foi criado também uma função apenas para exportar a matriz resultante do PDC: 'Criar_CSV_Vetores_RFinal_3DV2' e os parâmetros de entrada são análogos ao 'Criar_CSV_Apartir_Matriz_V2'.

O tipo de arquivo csv que é suportado pela solução utiliza o separador é o caractere ';' e não ','. Exemplo de um csv gerado pelo MatLab e que é suportado pelo RTMcarns:

```
1.6289785416510;1.1129008479445;0.5217839875905
1.4498802939959;1.1069412293169;-3.433931816664
1.2428078492499;-1.1346285704984;0.0303089816596
```

Este arquivo deve ser importado para uma matriz 3x3. Supondo que esse arquivo esteja na pasta local e o nome seja u_3x3.csv, o código a seguir exemplifica a utilização das funções de importação e exportação.

```
1 /*Declarar as variaveis*/
2 matrizV2 matriz_in,matriz_out;
3
4 /* Criar uma matriz 3x3 que recebera os valores do arquivo csv*/
5 matriz_in.lin = 3;
6 matriz_in.col = 3;
7 matriz_in.mat = (double**)( malloc(matriz_in.lin*sizeof(double*)) );
8 MLoc_Matriz(matriz_in.lin, matriz_in.col, matriz_in.mat);
9
10 /* Prencher a matriz criada com os valores que estao no csv */
11 Ler_Matriz_From_CSV_V2( &matriz_in , "./u_3x3.csv" );
12
13 /* Criar uma matriz 3x3 que armazenara o resultado */
14 matriz_out.lin = 3;
15 matriz_out.col = 3;
16 matriz_out.mat = (double**)( malloc(matriz_out.lin*sizeof(double*)) );
17 MLoc_Matriz(matriz_out.lin, matriz_out.col, matriz_out.mat);
18
19 /* Sob teste */
20 Inversa_NxN_V2( &matriz_in , &matriz_out );
21
22 /* Criando arquivo csv com a matriz resultante da funcao sob teste */
23 Criar_CSV_Apartir_Matriz_V2( &matriz_out , "./Resultado.csv" );
24
25 /* Liberando espaco alocado */
26 Free_Matriz(matriz_in.lin, matriz_in.col, matriz_in.mat);
27 Free_Matriz(matriz_out.lin, matriz_out.col, matriz_out.mat);
```

3.1.2 Processo de validação

Todas funções algébricas e algoritmos matemáticos presentes no projeto atual do RTMcarns foram validadas com o auxílio do software matemático MatLab. Observe na figura Figura 25 o processo utilizado para validação de uma função.

Funções que não foram detectadas erro numérico durante o processo de validação:

- Multiplicação de matrizes;
- Soma de matrizes;
- Subtração de matrizes;

- Determinante de uma matriz;
- Transposta da matriz;
- Produto de Kronecker;
- Rearranjo de matriz;
- E outras de igual simplicidade.

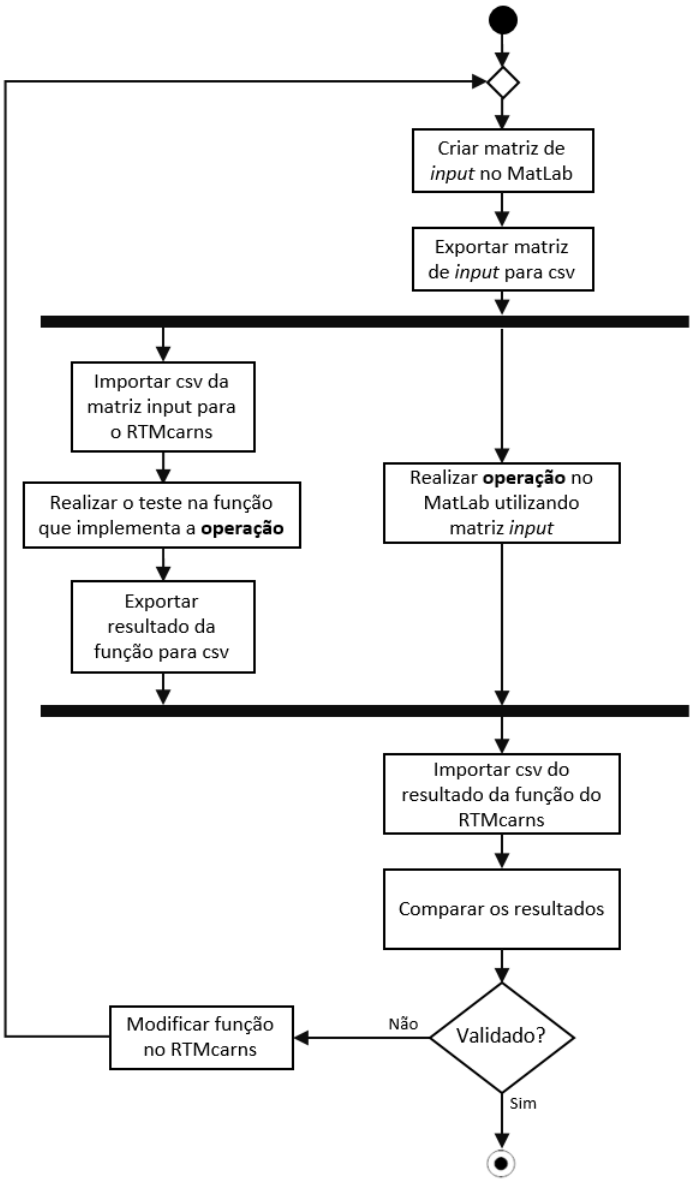

Figura 25: Procedimento de validação das funções

Porém o algoritmo de inversão de matriz possui diferenças de resultados em relação ao implementado pelo MatLab. Logo, todas as funções que dependem da operação de inversão também tiveram uma diferença em seus resultados frente ao MatLab.

Esse erro em relação aos resultados do Matlab possuem ordens de grandeza abaixo do que a aplicação exige. Verifique na Tabela 5 o erro médio e o desvio padrão da inversão de matrizes de tamanho 20x20, 200x200, 500x500 e 1000x1000. Essas matrizes foram geradas utilizando o comando **randi()** do MatLab, que basicamente cria uma matriz randômica com a dimensão estipulada via parâmetro. Sendo assim, existem grandes chances dessas matrizes serem mal condicionadas, podendo acarretar em maiores erros numéricos na inversão.

Tabela 5: Erro na inversão de matriz (utilizando *double*)

nxn	**Erro Médio**	s
$n = 20$	3,19E-15	3,18E-15
$n = 200$	2,17E-12	2,65E-12
$n = 500$	6,48E-11	6,87E-11
$n = 1000$	7,77E-10	9,28E-10

Foram utilizadas as equações (3.2) e (3.3) para construir a Tabela 5 e a Tabela 6.

$$E = |M - R| \tag{3.1}$$

$$e = \frac{\sum_i^n \sum_j^n |M_{i,j} - R_{i,j}|}{n^2} \tag{3.2}$$

$$s = \sqrt{\frac{\sum_{i,j}^n (E_{i,j} - e)^2}{n^2 - 1}} \tag{3.3}$$

Onde, M é a matriz do resultado da operação no MatLab, R é a matriz do resultado da operação no RTMcarns, n é a dimensão da matriz e s é o desvio padrão.

O RTMcarns utiliza ponto flutuante de dupla precisão (*double*). Observe na Tabela 6 como a mudança do tipo de precisão do ponto flutuante do sistema influencia no erro ao realizar uma inversão.

Tabela 6: Erro na inversão de matriz (*float*, *double* e *long double*)

	float		***double***		***long double***	
nxn	e	s	e	s	e	s
n = 20	1,43E-06	1,58E-06	3,19E-15	3,18E-15	1,97E-17	2,29E-17
n = 200	0,0046	0.0059	2,17E-12	2,65E-12	5,39E-16	5,93E-16
n = 500	0,0029	0,0027	6,48E-11	6,87E-11	4,18E-14	4,68E-14
n = 1000	0,005	0.0054	7,77E-10	9,28E-10	4,20E-13	5,0087E-13

A Tabela 6 mostra que a precisão numérica simples (*float*) não é suficiente para a implementação desse tipo de algoritmo, porém o erro médio para a precisão do tipo *double*

é o suficiente para a maioria das aplicações que envolvem EEG, dado que os melhores sistemas de captação de sinal de EEG possuem resolução máxima entre 12 e 16 bits, desconsiderando o ruído (RAMPIL; IRA, 1998).

3.1.3 Precisão do algoritmo de PDC

Para o teste de precisão a seguir foi utilizado um sinal u_{5x2000} que possui correlação entre os canais e foi gerado através das equações (3.4), (3.5), (3.6), (3.7) e (3.8).

Observe pelas equações que o sentido das correlações são: $x_1 \rightarrow x_2$, $x_2 \rightarrow x_3$, $x_1 \rightarrow x_4$, $x_3 \rightarrow x_4$, $x_4 \rightarrow x_5$ e fechando em $x_5 \rightarrow x_2$.

$$x_1(t) = 0,95\sqrt{2}.x_1(t-1) - 0,9025.x_1(t-2) + w_i(1,t) \tag{3.4}$$

$$x_2(t) = -0,5.x_1(t-1) + 0,5.x_5(t-2) + w_i(2,t) \tag{3.5}$$

$$x_3(t) = 0,4.x_2(t-2) + w_i(3,t) \tag{3.6}$$

$$x_4(t) = -0,5.x_3(t-1) + 0.25\sqrt{2}.x_4(t-1) + 0,25.\sqrt{2}.x_5(t-1) + w_i(4,t) \tag{3.7}$$

$$x_5(t) = -0,25\sqrt{2}.x_4(t-1) + 0,25\sqrt{2}.x_5(t-1) + w_i(5,t) \tag{3.8}$$

Onde w_i é uma matriz randômica com distribuição normal.

Visualize a seguir o trecho de um *script* em MatLab que gera o sinal u_{5x2000} a partir das relações acima.

```
ndiscard = 1000; % number of points discarded at beginning of series
ns = 2000;   % number of analyzed samples points
nd = ns + ndiscard;

randn('state', 3.090663000000000e+003)

wi = randn(5,nd);
x1 = zeros(1,nd); x2 = zeros(1,nd); x3 = zeros(1,nd); x4 = zeros(1,nd);
x5 = zeros(1,nd);
% Variables initialization
for t = 1:4,
x1(t) = randn(1); x2(t) = randn(1); x3(t) = randn(1); x4(t) = randn(1);
x5(t) = randn(1);
```

```
end;

for t=5:nd,
x1(t) = 0.95*sqrt(2)*x1(t-1) - 0.9025*x1(t-2)+ wi(1,t);
x2(t) = -0.5*x1(t-1) + 0.5*x5(t-2) + wi(2,t); % The loop is
%                      ~~~~~~~~~~              % closed from x5 -> x2
x3(t) = 0.4*x2(t-2) + wi(3,t);
x4(t) = -0.5*x3(t-1) + 0.25*sqrt(2)*x4(t-1) + 0.25*sqrt(2)*x5(t-1) ...
+ wi(4,t);
x5(t) = -0.25*sqrt(2)*x4(t-1) + 0.25*sqrt(2)*x5(t-1) + wi(5,t);
end;

y = [x1' x2' x3' x4' x5'];
u = y(nd - ns + 1:nd,:);
u = u'
```

O formato do sinal gerado pode ser observado na Figura 26, em que o sinal u possui cinco canais com duas mil amostras cada.

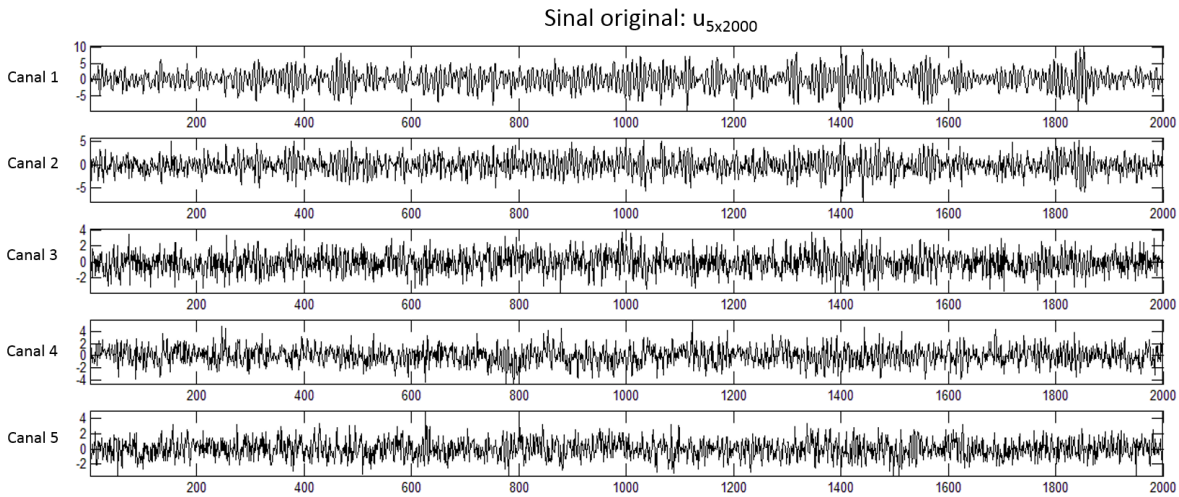

Figura 26: Sinal gerado para teste de precisão numérica

Para realizar o teste de precisão de todo o algoritmo do RTMcarns que, tem por objetivo calcular a coerência parcial direcionada, seguiu-se o processo da Figura 25 utilizando variações de sinais a partir de u.

Foram utilizados três sinais: um com apenas três canais de u, outro com quatro e o sinal original, u_{3x2000}, u_{4x2000} e u_{5x2000}, respectivamente.

Para montar a Tabela 7 foi realizado o cálculo de PDC para cada um dos sinais descritos acima, variando a ordem do modelo de regressão linear entre 6, 12 e 18 além das equações (3.2) e (3.3).

Todos os resultados contidos na Tabela 7 foram obtidos com o sistema utilizando *double* como base de precisão numérica.

Tabela 7: Erro do cálculo da Coerência Parcial Direcionada

	ordem = 6		***ordem = 12***		***ordem = 18***	
nxn	e	s	e	s	e	s
u_{3x2000}	7,98E-16	1,42E-15	8,15E-16	1,35E-15	8,21E-16	1,34E-15
u_{4x2000}	7,88E-16	1,33E-15	8,10E-16	1,31E-15	8,05E-16	1,30E-15
u_{5x2000}	3,60E-16	5,18E-16	3,70E-16	4,99E-16	3,73E-16	4,90E-16

Para demonstrar os resultados obtidos, será apresentado dados da execução do algoritmo Mcarns e PDC para o sinal u_{5x2000} com ordem =6.

O sinal e a ordem são os parâmetros de entrada para o Mcarns, que irá devolver as matrizes com os coeficientes autorregressivos multivariado do modelo. Para a execução descrita seguem em (3.9), (3.10), (3.11), (3.12), (3.13) e (3.14) os coeficientes do modelo.

$$A_1 = \begin{bmatrix} 1,3589 & -0,0201 & 0,0082 & 0,0334 & 0,0184 \\ -0,4791 & -0,0050 & 0,0026 & 0,0008 & -0,0156 \\ 0,0236 & 0,0034 & -0,0151 & -0,0045 & 0,0180 \\ 0,0371 & -0,0265 & -0,5192 & 0,3850 & 0,3711 \\ -0,0330 & 0,0291 & -0,0130 & -0,3633 & 0,3497 \end{bmatrix} \tag{3.9}$$

$$A_2 = \begin{bmatrix} -0,9203 & -0,0207 & 0,0384 & 0,0338 & -0,0180 \\ -0,0071 & 0,0362 & -0,0175 & -0,0188 & 0,5043 \\ -0,0218 & 0,4170 & 0,0568 & -0,0096 & -0,0180 \\ -0,0667 & 0,0055 & 0,0090 & -0,0467 & -0,0092 \\ 0,0290 & 0,0082 & -0,0001 & 0,0767 & -0,0155 \end{bmatrix} \tag{3.10}$$

$$A_3 = \begin{bmatrix} 0,0016 & 0,0008 & 0,0231 & -0,0360 & 0,0146 \\ 0,0005 & -0,0226 & 0,0224 & 0,0015 & 0,0194 \\ 0,0270 & 0,0189 & -0,0310 & -0,0477 & 0,0226 \\ -0,0179 & 0,0455 & -0,0072 & 0,0080 & 0,0456 \\ 0,0079 & -0,0409 & 0,0223 & -0,0132 & -0,0338 \end{bmatrix} \tag{3.11}$$

$$A_4 = \begin{bmatrix} -0,0035 & -0,0268 & -0,0223 & -0,0201 & 0,0154 \\ -0,0067 & -0,0151 & -0,0246 & 0,0157 & -0,0146 \\ -0,0026 & 0,0183 & -0,0375 & 0,0248 & -0,0122 \\ 0,0756 & 0,0109 & -0,0426 & 0,0665 & -0,0040 \\ -0,0161 & -0,0401 & 0,0249 & 0,0144 & -0,0645 \end{bmatrix} \tag{3.12}$$

$$A_5 = \begin{bmatrix} 0,0186 & 0,0113 & -0,0238 & 0,0071 & 0,0167 \\ 0,0283 & -0,0317 & 0,0052 & 0,0220 & 0,0046 \\ 0,0331 & 0,0245 & 0,0018 & -0,0130 & 0,0154 \\ -0,0176 & -0,0136 & 0,0359 & -0,0500 & -0,0142 \\ -0,0456 & 0,0394 & 0,0329 & -0,0398 & 0,0321 \end{bmatrix} \tag{3.13}$$

$$A_6 = \begin{bmatrix} -0,0029 & 0,0220 & -0,0051 & -0,0055 & -0,0172 \\ -0,0416 & 0,0089 & 0,0310 & -0,0257 & -0,0007 \\ 0,0144 & 0,0205 & 0,0053 & -0,0137 & -0,0221 \\ -0,0332 & 0,0206 & -0,0164 & 0,0067 & 0,0240 \\ 0,0327 & -0,0199 & -0,0329 & -0,0052 & 0,0272 \end{bmatrix} \tag{3.14}$$

Sendo (3.15) a matriz de covariância:

$$\sigma = 10^{-3}. \begin{bmatrix} 1,8296 & 0,0677 & -0,0137 & -0,0008 & 0,0734 \\ 0,0677 & 1,9046 & -0,0110 & -0,0838 & 0,0308 \\ -0,0137 & -0,0110 & 1,9528 & 0,0118 & -0,0104 \\ -0,0008 & -0,0838 & 0,0118 & 2,0139 & -0,0047 \\ 0,0734 & 0,0308 & -0,0104 & -0,0047 & 1,9408 \end{bmatrix} \tag{3.15}$$

Essas matrizes são as entradas da função que calcula o PDC. Os coeficientes autorregressivos multivariados (A_p) são levados ao domínio da frequência, onde o PDC é calculado.

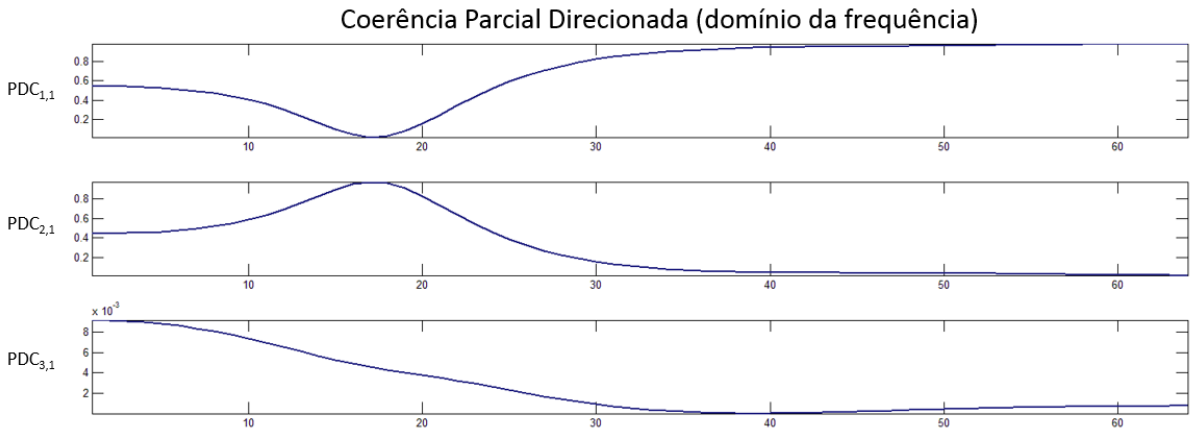

Figura 27: Gráfico referentes a $PDC_{1,1}$, $PDC_{2,1}$ e $PDC_{3,1}$

Como mencionado em tópicos anteriores, o PDC resulta em uma matriz tridimensional, em que cada um dos vetores formados na terceira dimensão são valores no domínio discreto da frequência, possuindo $\lambda = 64$ raias equiespaçadas de 0 a 0,5. Observe na

Figura 27 o gráfico dos três primeiros vetores do PDC e a lista completa com todos os gráficos formados pelos vetores da terceira dimensão estão no anexo A.

3.2 Performance do sistema

Para mensurar o desempenho do RTMcarns foram criados dois cenários de teste:

1. Algoritmo MatLab x RTMcarns executando no PC
2. RTMcarns PC x RTMCarns Embarcado

O objetivo do primeiro teste é verificar o desempenho da implementação C do RTMcarns contra a do mesmo algoritmo executando diretamente do MatLab. Sendo assim eles devem executar na mesma plataforma para que os resultados façam sentido.

No segundo teste a intensão é verificar o desempenho real do sistema na plataforma embarcada e a comparação com o resultado do PC é apenas para ter uma referência e mensurar a diferença de desempenho das plataformas frente a uma mesma implementação.

Com os resultados do sistema executando na Wand Board, pode-se definir restrições de intervalo mínimo entre os ciclos de cálculo, segundo as características do sinal e as configurações do usuário.

3.2.1 Teste comparativo no computador

3.2.1.1 Ambiente de teste

O RTMcarns, como todo software, é executado dentro de um processo no sistema operacional, e a fatia de tempo fornecida pelo SO para o RTMcarns é distribuída entre as *threads* existentes.

Observe na Figura 3 do capítulo anterior, a visão simplificada do sistema com suas *threads*. Quando é iniciado um cálculo utilizando os algoritmos de Mcarns e PDC no Módulo de Cálculo, ela pode ser interrompida a qualquer momento pelo escalonador para que outras *threads* executem.

Dado que esse é um teste comparativo, a solução implementada em MatLab que calcula o PDC é executada sem interrupções de outras *threads* do próprio processo. Então foi gerado uma versão do RTMcarns com apenas o Módulo de Cálculo ativo para que não haja interrupção ao iniciar o algoritmo. Sendo assim, será testado a eficiência do Módulo de Cálculo e das funções de álgebra linear implementadas em C frente a solução existente no MatLab.

Observe na Figura 28 a versão modificada do sistema criado para realizar este teste.

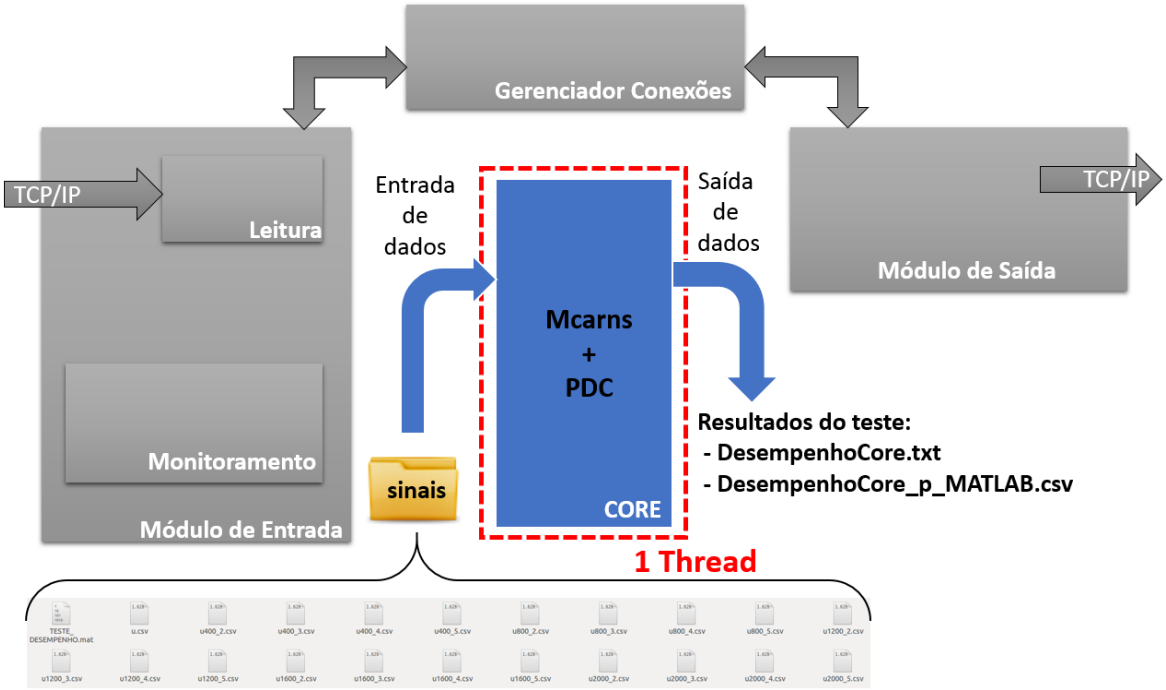

Figura 28: Ambiente de teste *mono thread*

Basicamente, o Módulo de Cálculo realiza a leitura de arquivos csv de uma pasta local chamada 'sinais', cria as matrizes de entrada para cada um dos testes, realiza o cálculo do PDC e registra a quantidade de tempo utilizado no cálculo.

Para cada combinação sinal/ordem o cálculo é executado dez vezes, tirado a média de desempenho e então registrado em arquivos locais. Para registrar os resultados de desempenho de cada execução são gerados dois tipos de arquivos.

- DesempenhoCore_p_MATLAB: são os resultados em um formato csv para serem importados no MatLab para análise e comparação;
- DesempenhoCore.txt: possui informações de cada execução em um formato simples para leitura. Verifique no anexo B o arquivo gerado para um teste. Foram colocadas apenas duas páginas para demonstrar o formato do arquivo, sendo que o resultado completo possui 1205 linhas.

O mesmo computador foi utilizado para executar os testes. Segue a configuração do computador utilizado.

- **Processador**: Intel i7 4770 3,40 GHz (quatro núcleos);
- **Memória RAM**: 16 GB DDR3 (2133 MHz);
- **Placa Gráfica**: GeForce GTX 770 4 GB;

- **Sistema Operacional**:

  Windows 64 Bits para teste com o MatLab;

  Linux Ubuntu 12.04 64 Bits para RTMcarns.

3.2.1.2 Amostras de dados do teste

Os arquivos csv foram criados a partir do sinal u_{5x2000}, apresentado no tópico 3.1.3. Foram criados sinais com 2, 3, 4 e 5 canais e em cada um deles existe uma variação com 400, 800, 1200, 1600 e 2000 amostras, totalizando 20 sinais diferentes:

- u_{2x400}, u_{2x800}, u_{2x1200}, u_{2x1600} e u_{2x2000};
- u_{3x400}, u_{3x800}, u_{3x1200}, u_{3x1600} e u_{3x2000};
- u_{4x400}, u_{4x800}, u_{4x1200}, u_{4x1600} e u_{4x2000};
- u_{5x400}, u_{5x800}, u_{5x1200}, u_{5x1600} e u_{5x2000}.

Para cada um desses sinais o algoritmo completo foi executado variando a ordem do modelo de regressão para: 3, 6, 9, 12, 15, 18, 21, 24, 27 e 30. Este cenário gera uma combinação de 200 cenários, e cada foram realizados 10 testes e armazenados, juntamente com a média de tempo de execução em cada cenário.

3.2.1.3 Resultados

Este trabalho não tem a intuito de ter maior performance que o MatLab. O interesse é uma implementação sólida em C, que possua resultados tão precisos quanto. O mais importante é que o sistema consiga coletar dados e realizar cálculos continuamente e que fosse feito em tempo real, além de oferecer uma plataforma para estudos de algoritmos que torne o cálculo de conectividade neural em tempo real uma realidade cada vez mais viável.

O sistema foi compilado em 4 níveis de otimização diferentes:

- -O0: sem otimização;
- -O1: nível 1 de otimização;
- -O2: nível 2 de otimização;
- -O3: nível 3 de otimização.

O desempenho do RealTimeMcarns compilado sem otimização, em média, é pior que a do MatLab, principalmente quando aumenta o número de amostras de cada canal. Ao ligar a otimização do compilador (GNU) o desempenho do RTMcarns melhora a medida

que aumenta-se o nível de otimização, porém o tamanho do código compilado e o uso de memória aumentam também.

Os teste de performance mensurou o tempo de execução dos dois grandes algoritmos separadamente, será mostrado a seguir que o tempo de execução do Mcarns possui uma ordem de grandeza maior que a função que calcula o PDC.

Na Figura 29 e Figura 30 estão apresentados os gráficos comparativos entre o desempenho do Mcarns no MatLab e no RTMarns com cada uma das otimizações. Serão exibidos os resultados dos gráficos com 800 e 1600 amostras, o teste completo com todos gráficos está disponível no anexo C.

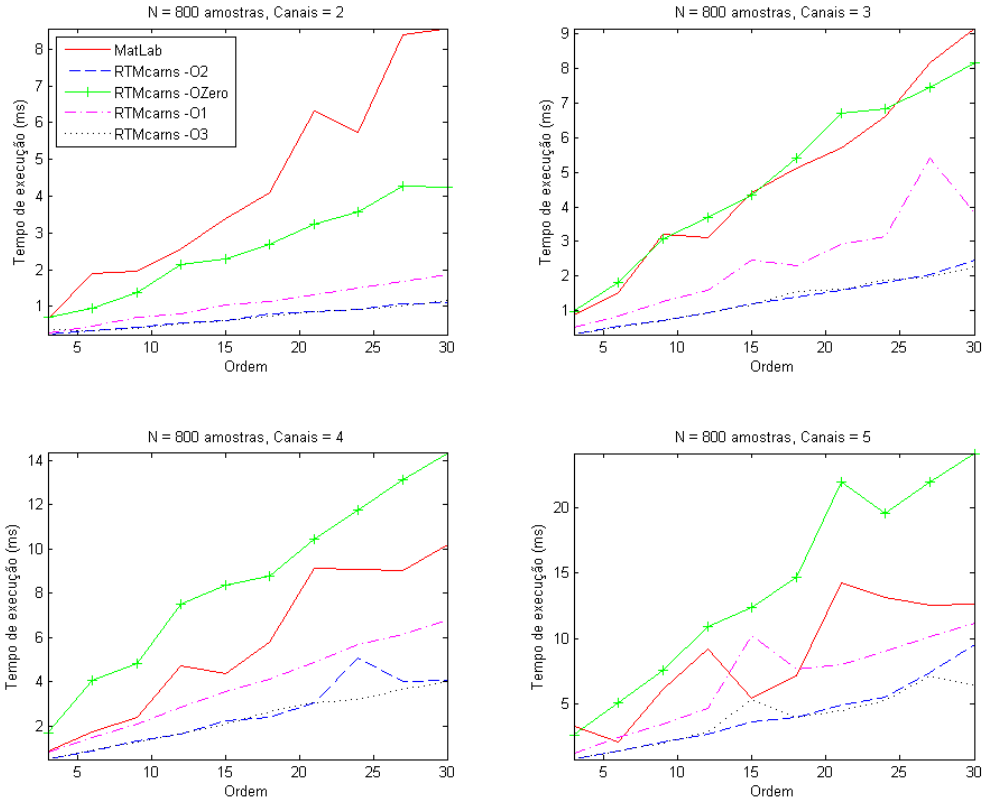

Figura 29: Desempenho da função Mcarns do MatLab x RTMcarns todas otimizações para u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

Observe que em todos os testes o RTMCarns com otimizações níveis 2 e 3 possuem desempenho melhor que o do MatLab, porém pode-se verificar que, a medida que o número de canais aumenta o RTMcarns piora seu desempenho mais que o MatLab, fazendo com que as linhas de desempenho -O2 e -O3 se aproximem da linha do MatLab. A tendência é que ao continuar aumentando o número de canais do sinal, o desempenho do MatLab se torne melhor que de qualquer compilação do RTMcarns.

É provável que esse comportamento se atribua a implementação da função de inversão de matriz, pois o aumento na quantidade de canais influencia diretamente no

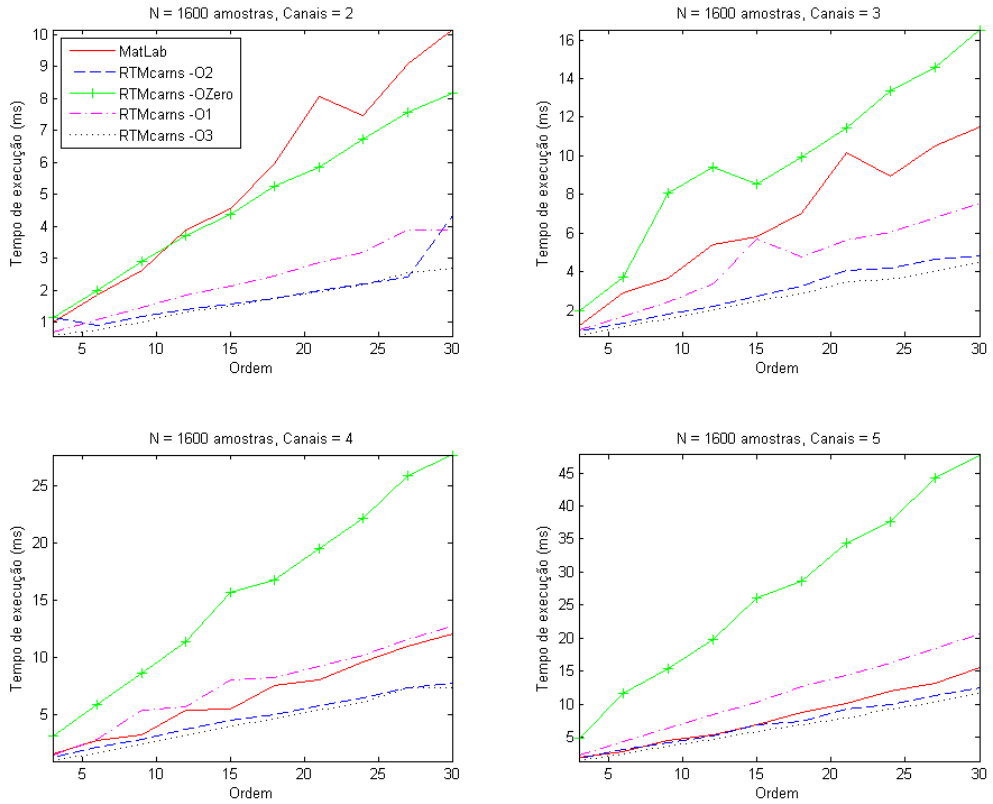

Figura 30: Desempenho da função Mcarns do MatLab x RTMcarns todas otimizações para u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

tamanho da matriz a ser invertida pela função Mcarns.

Nos gráficos 31, 32, 33, 34 pode-se observar que a diferença do tempo de execução do Mcarns é de pelo menos dez vezes a do PDC. Para essas comparações foram deixados no gráfico apenas o desempenho do MatLab e do RTMcarns -O3. Na Figura 31 a execução do Mcarns e na Figura 32 o PDC para o mesmo sinal e mesmo ciclo de cálculo. O mesmo vale para a Figura 33 e Figura 34. Os gráficos de todos os sinais da função PDC encontram-se no anexo D.

3.2.2 Teste no Embarcado

Neste tópico será mostrado o desempenho do RTMcarns executando diretamente da placa de desenvolvimento Wand Board Quad mostrado na sessão 2.3. Nesse teste foi utilizado o mesmo conjunto de sinais da sessão 3.2.1.2. Cada uma das combinações de sinal/ordem foram executadas 10 vezes, assim como o teste no computador.

Devido a arquitetura do ARM embarcado na Wand Board ser 32 bits, é esperado uma queda significativa no desempenho se comparado com o computador. E além disso o ARM estará executando em um *clock* de 1 GHz frente a 3.4 GHz do computador. Observe na Figura 35 e Figura 36 a diferença de desempenho entre o computador e o ambiente

embarcado, para os sinais de 800 e 1600 amostras para calcular a função Mcarns. Todos os gráficos estão disponíveis no anexo E.

Nesses testes ambos os RTMcarns estão compilados com otimização nível 3.

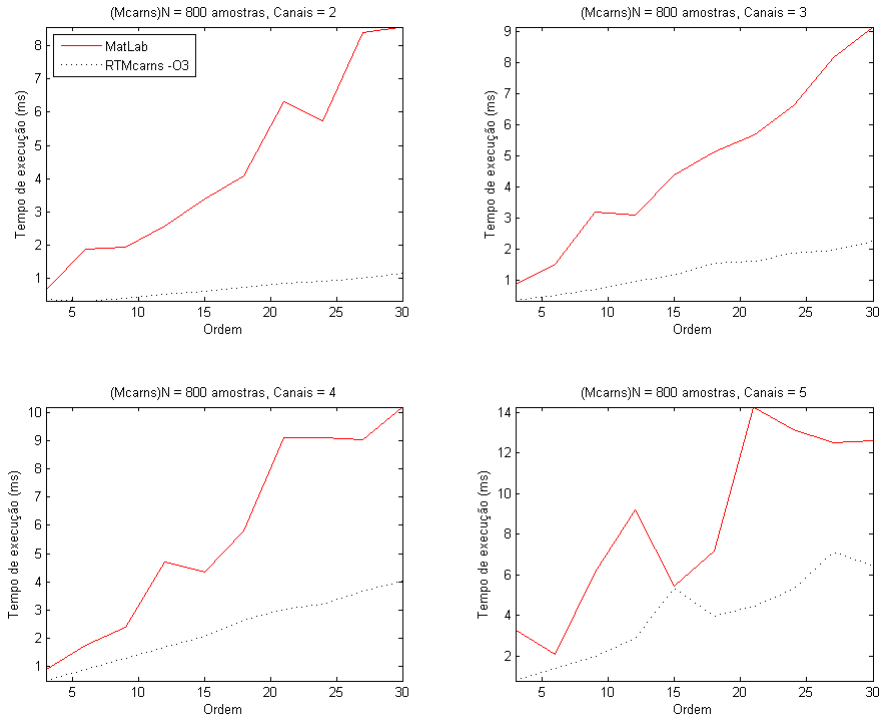

Figura 31: Desempenho da função Mcarns do MatLab x RTMcarns -O3 para u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

A diferença de desempenho é de aproximadamente 14 vezes. E a partir dos resultados obtidos pode-se calcular as limitações de tempo real do sistema.

3.2.2.1 Limitações do sistema

A partir dos testes realizados na placa de desenvolvimento é possível calcular o intervalo mínimo que deve existir entre os ciclos de cálculo.

Para encontrar a fórmula do intervalo mínimo foram utilizadas aproximações lineares das curvas de desempenho de cada canal, verifique na Figura 37 gráficos do número de amostras (N) $\times$ tempo em milissegundos de cada canal, para ordem p=3. As informações contidas na Figura 37 ajudaram a encontrar a relação de N com o tempo gasto e também a razão da inclinação entre cada uma das funções dos canais.

No segundo gráfico, contido na Figura 38, estão as informações de desempenho para ordem p=6, que ajudaram a encontrar a razão entre os equações da Figura 37 e Figura 38, parte fundamental para entender como a ordem influencia no desempenho total do sistema.

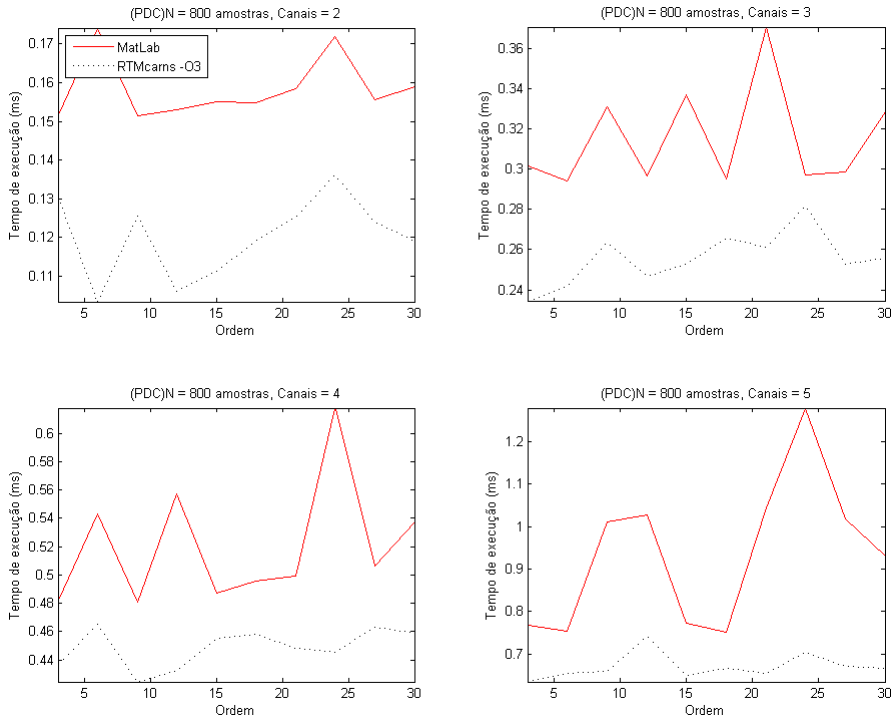

Figura 32: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

Na equação (3.16) é apresentado uma aproximação a partir dos dados coletados nos testes, e que servirá de referência para conhecer o intervalo mínimo que o sistema demorará para realizar um ciclo de cálculo segundo: o número de canais do sinal, o número de amostras de cada canal e a ordem do modelo autorregressivo.

$$T = (0,227p + 0,33)(1,6^{(N_c-2)}(0,0044N - 0,189) + 0,23) \tag{3.16}$$

em que, p é a ordem do modelo autorregressivo, N é o número de amostras e N_c

Exemplo de estimação: Dado que o RTMcarns foi ajustado para realizar o cálculo em "batelada"de um sinal de 14 canais com 1000 amostras, e a ordem do estimador do modelo é p= 10.

Então, tem-se $N_c = 14$, $N = 1000$ e $p = 10$, o tempo mínimo que o RTMcarns levaria para calcular é aproximadamente 3116 milissegundos, ou seja, 3,1 segundos. Então cabe ao usuário decidir se esse intervalo entre os ciclos de cálculo é o suficiente ou não para sua aplicação.

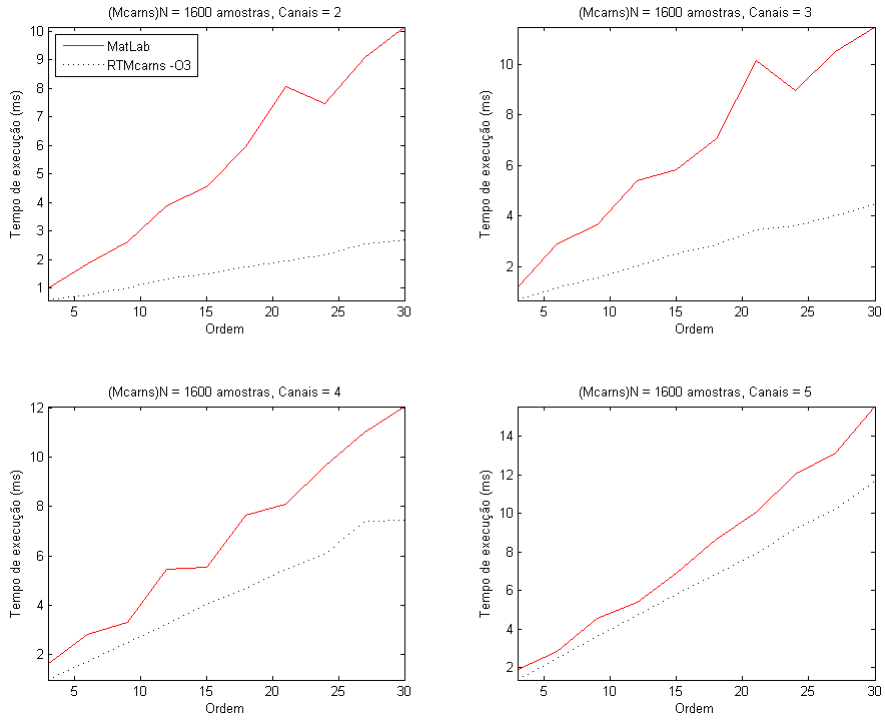

Figura 33: Desempenho da função Mcarns do MatLab x RTMcarns -O3 para u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

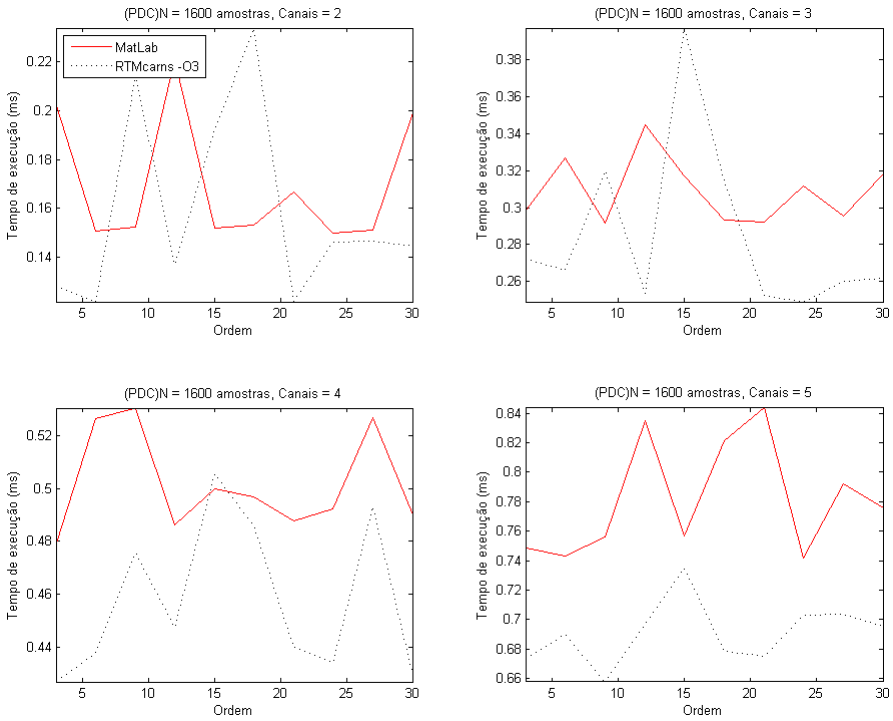

Figura 34: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

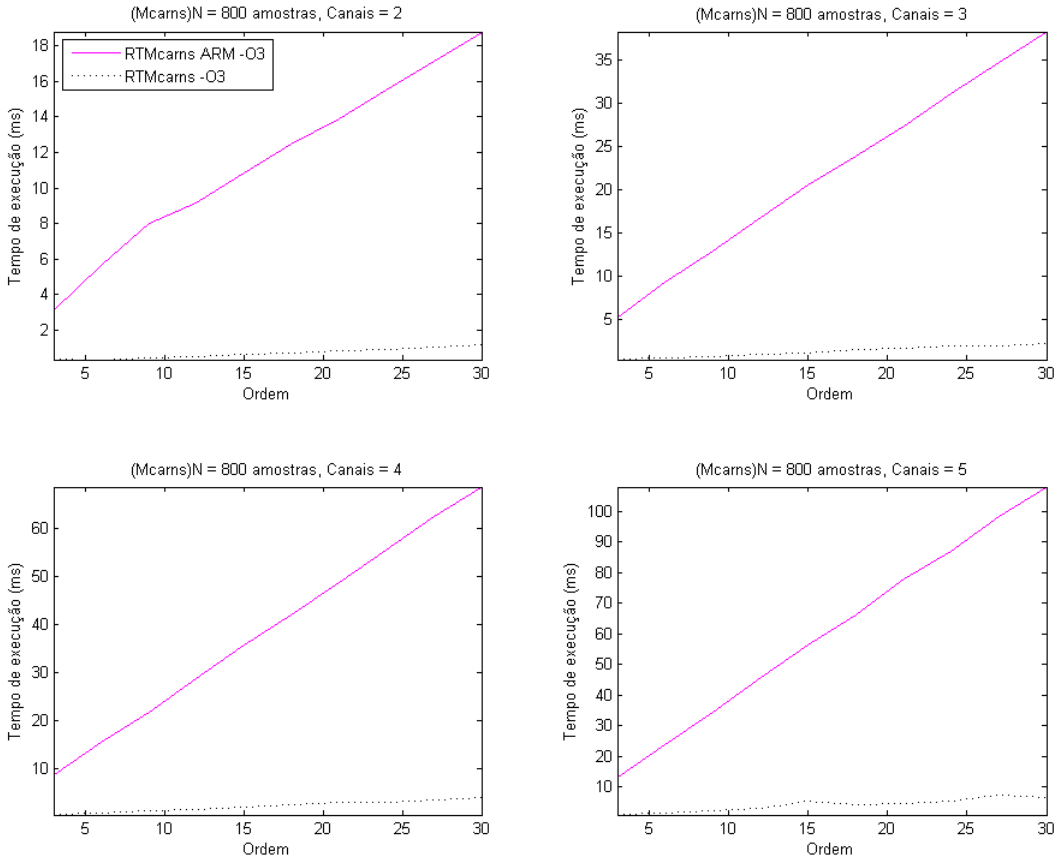

Figura 35: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

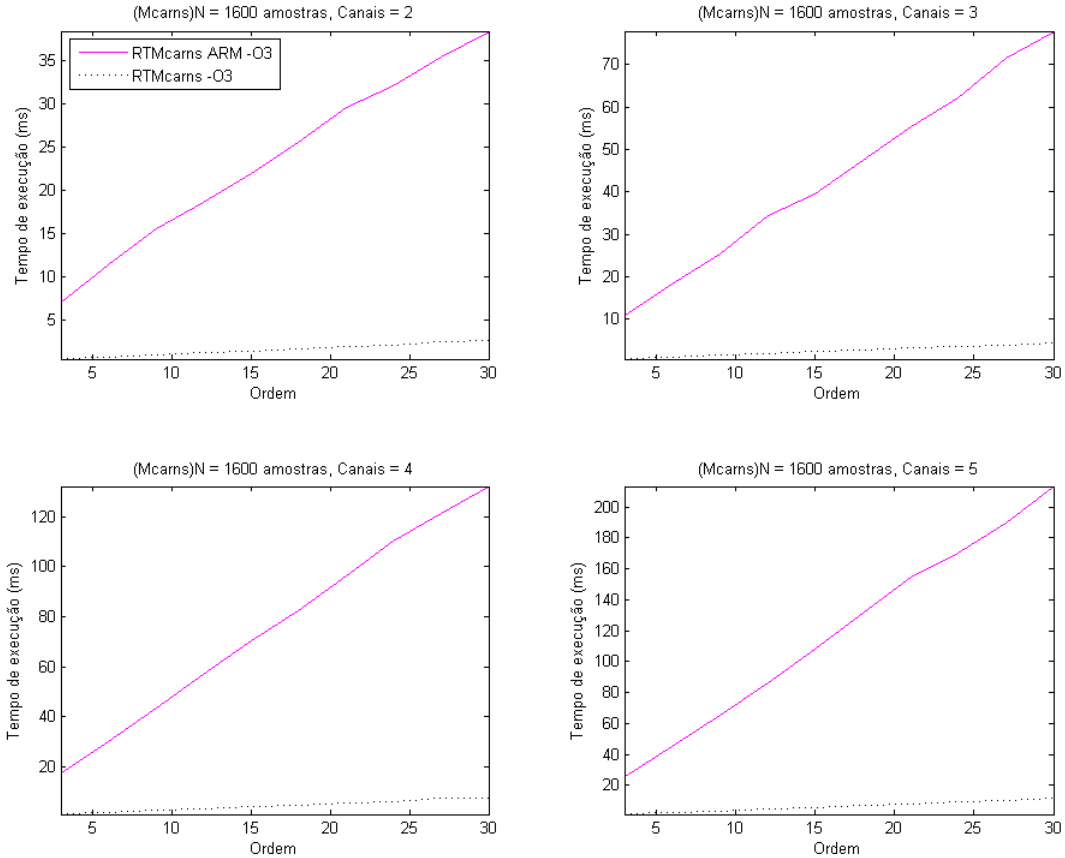

Figura 36: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

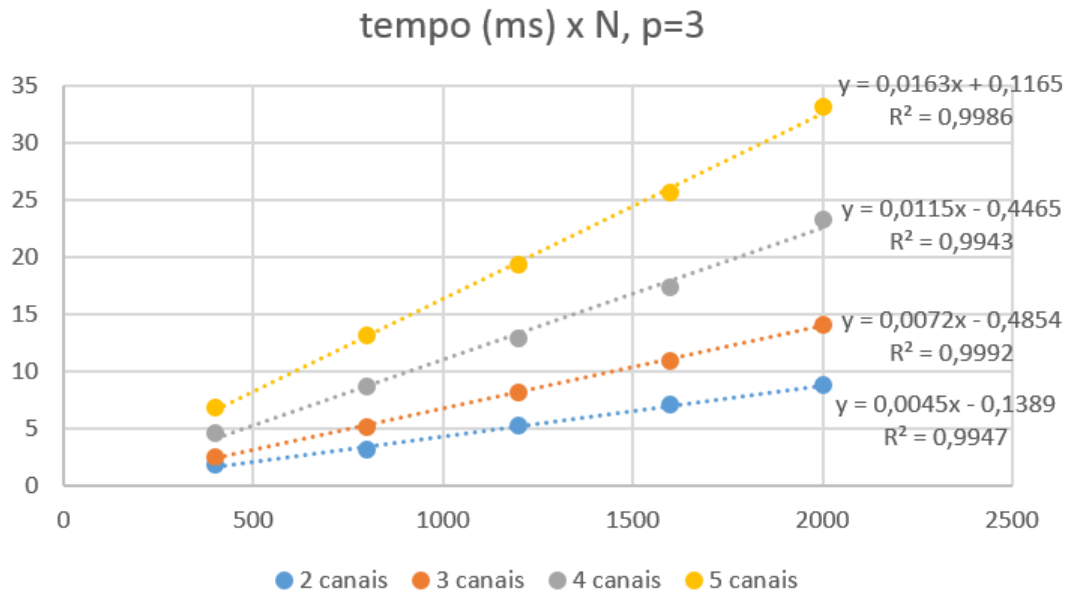

Figura 37: Curvas de desempenho entre os canais para p=3

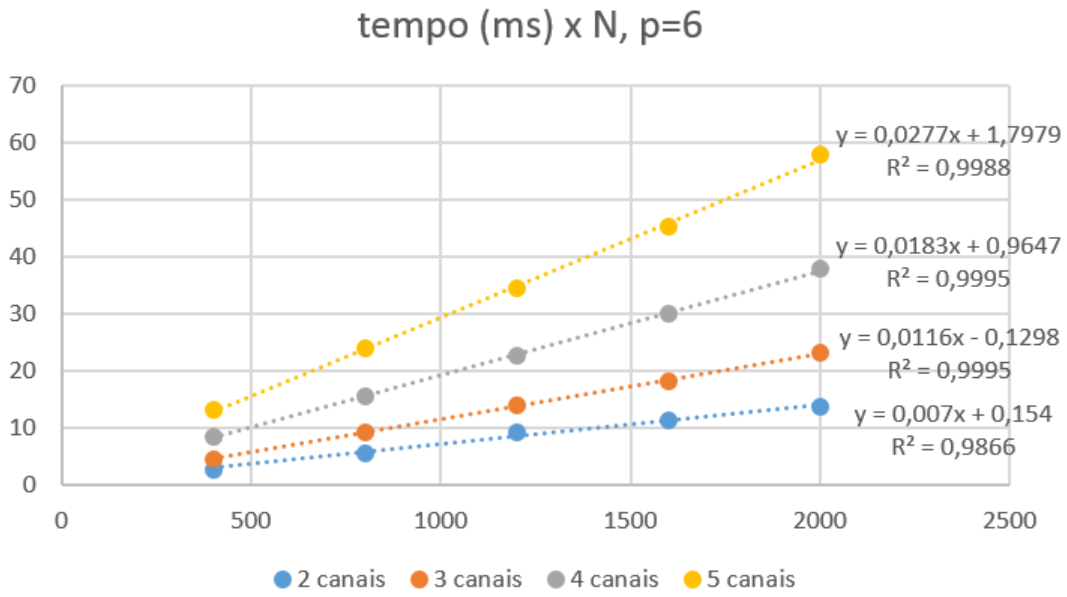

Figura 38: Curvas de desempenho entre os canais para p=6

4 Conclusões

Este trabalho reconheceu a importância de colaborar com o avanço da medicina na investigação e entendimento de detalhes do funcionamento do cérebro. A partir desse momento, entendeu-se a necessidade de um dispositivo que permitisse estudos na interrelação entre as regiões cerebrais, e possibilitasse que, mesmo para um corpo médico sem conhecimento em *softwares* matemáticos, obtivesse resultados e índices precisos.

Criou-se então um plataforma embarcada que possibilita a mobilidade dos experimentos, que realiza os cálculos continuamente e em tempo real, sem a necessidade de softwares *matemáticos* especializados. Sabendo que criar uma implementação ótima logo de início não seria possível, decidiu-se por implementar um sistema que possibilitasse a evolução e melhoria, tanto em novos métodos para se calcular a conectividade neural, melhorando a implementação já existente ou modificando o *hardware* onde está implementado.

O projeto atual atendeu as expectativas de criar uma primeira implementação de tempo real respeitando certos limites de tempo entre os ciclos dependendo das características do sinal, que certamente não existirão em futuras implementações do RTMcarns.

A precisão numérica foi um ponto atingido com satisfação neste trabalho, sendo ainda mais importante que as restrições de tempo do sistema, já que isso influi diretamente na confiabilidade do sistema e por sua vez na sua adoção como referência diagnóstica.

É esperado que a plataforma incentive novos engenheiros e cientistas a estudar e trabalhar com modelos e implementações de tempo real para conectividade neural. É igualmente esperado que a medicina demonstre mais interesse nos indicativos de conectividade neural para seus estudos, já que essas novas plataformas visam facilitar cada vez mais obtenção dos resultados.

4.1 Trabalhos futuros

Como trabalhos futuros pode-se considerar o uso da placa gráfica embarcada no dispositivo para melhorar o desempenho das operações matriciais. Existem bibliotecas já portadas para o ARM que facilitam esse tipo de desenvolvimento, como a ArrayFire <http://arrayfire.com/> acesso em: 01/02/2016.

Desenvolver um hardware dedicado para o cálculo do PDC utilizando FPGA seria uma outra vertente de trabalho e uma aposta no altíssimo desempenho que o sistema pode obter utilizando esse tipo de solução. Sendo que é possível embarcar processadores na FPGA e criar coprocessadores dedicados a operações matriciais mais árduas.

E por último mas não menos importante, a introdução de um componente gráfico ao sistema, que gere gráficos e indicadores amigáveis para um corpo médico. Seja esse gráfico gerado internamente no sistema embarcado ou por meio de um software complementar que se conectado na saída do RTMcarns.

Referências

AKAIKE, H. A new look at statistical model identification. *IEEE Trans. Autom. Control*, v. 19, p. 716–723, 1974. Citado na página 23.

BACCALA, L. A. et al. Unified asymptotic theory for all partial directed coherence forms. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, v. 371, n. 1997, p. 1–13, 2013. Citado 2 vezes nas páginas 17 e 22.

BACCALA, L. A.; SAMESHIMA, K. Overcoming the limitations of correlation analysis for many simultaneously processed neural structures. *Progress in Brain Research, Advances in Neural Population Coding*, v. 130, p. 33–47, 2001. Citado 2 vezes nas páginas 17 e 21.

BACCALA, L. A.; SAMESHIMA, K. Partial directed coherence: a new concept in neural structure determination. *Biol. Cybern.*, v. 84, n. 6, p. 463–474, 2001. Citado 2 vezes nas páginas 17 e 22.

BASAR, E. *Memory and Brain Dynamics: Oscillations Integrating Attention, Perception, Learning, and Memory*. [S.l.]: CRC Press, 2004. Citado na página 22.

BROCKWELL, P. J.; DAVIS, R. A. *Time Series: Theory and Methods*. [S.l.]: Springer, 1987. Citado 2 vezes nas páginas 20 e 22.

BURNHAM, K. P.; ANDERSON, D. R. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. [S.l.]: Springer, 2003. Citado na página 23.

BUZSÁKI, G. *Rhythms of the Brain*. [S.l.]: Oxford University Press, 2006. Citado na página 22.

COOTER, R. *The Cultural Meaning of Popular Science: Phrenology and the Organization of Consent in Nineteenth-Century Britain*. [S.l.]: Cambridge University Press, 1984. Citado na página 17.

CORMEN, T. *Introduction to Algorithms*. MIT Press, 2009. ISBN 9780262533058. Disponível em: <https://books.google.com.br/books?id=aefUBQAAQBAJ>. Citado 2 vezes nas páginas 62 e 64.

COTTLE, R. W. Manifestations of the schur complement. *Linear Algebra and its Applications*, v. 8, n. 3, p. 189 – 211, 1974. ISSN 0024-3795. Disponível em: <http://www.sciencedirect.com/science/article/pii/0024379574900664>. Citado na página 62.

DAVIE, A. M.; STOTHERS, A. J. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, v. 143, p. 351–369, 4 2013. ISSN 1473-7124. Disponível em: <http://journals.cambridge.org/article_S0308210511001648>. Citado na página 64.

DUAN, G.-R. *Generalized Sylvester Equations: Unified Parametric Solutions*. [S.l.]: CRC Press, 2015. Citado na página 25.

FOUNDATION, N. *Drug Resistance in Epilepsy: Lessons From Oncology - No. 243*. [S.l.]: Wiley, 2002. Citado na página 18.

FREEMAN, W.; QUIROGA, R. Q. *Imaging Brain Function With EEG: Advanced Temporal and Spatial Analysis of Electroencephalographic Signals*. [S.l.]: Springer, 2012. Citado na página 17.

GRANGER, C. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, v. 37, p. 424–438, 1969. Citado 3 vezes nas páginas 6, 7 e 20.

HAYKIN, S. *Adaptive Filter Theory*. [S.l.]: Pearson, 2011. Citado na página 23.

KAMINSKI, M. J.; BLINOWSKA, K. J. A new method of the description of the information flow in the brain structures. *Biol. Cybern.*, v. 65, p. 203–210, 1991. Citado na página 17.

LEE-CHIONG, T. *Sleep Medicine: Essentials and Review*. [S.l.]: Oxford University Press, 2008. Citado na página 18.

LI, X. *Functional Magnetic Resonance Imaging Processing*. [S.l.]: Springer, 2013. Citado na página 17.

LüTKEPOHL, H. *Handbook of Matrices*. [S.l.]: Chichester ; Wiley, c1996., 1996. v. 1. Citado na página 25.

LüTKEPOHL, H. *New Introduction to Multiple Time Series Analysis*. [S.l.]: Springer, New York, 2005. Citado 3 vezes nas páginas 20, 22 e 23.

MARPLE, S. L. *Digital Spectral Analysis: With Applications (Prentice-Hall Series in Signal Processing)*. [S.l.]: Prentice Hall, 1987. ISBN 0132141493. Citado 2 vezes nas páginas 23 e 25.

MATHEW, J. et al. *Eco-friendly Computing and Communication Systems: International Conference, ICECCS 2012, Kochi, India, August 9-11, 2012. Proceedings*. Springer Berlin Heidelberg, 2012. (Communications in Computer and Information Science). ISBN 9783642321122. Disponível em: <https://books.google.com.br/books?id=A--5BQAAQBAJ>. Citado na página 62.

MCQUARRIE, A. D. R.; TSAI, C. ling. *Regression and Time Series Model Selection*. [S.l.]: World Scientific, 1998. Citado na página 23.

MITRA, P.; BOKIL, H. *Observed Brain Dynamics*. [S.l.]: Oxford University Press US, 2008. Citado na página 17.

MOLLER, E. et al. Instantaneous multivariate EEG coherence analysis by means of adaptive high-dimensional autoregressive models. *Journal of neuroscience methods*, v. 105, n. 2, p. 143–158, 2001. Citado na página 23.

MYERS, R.; CUNNINGHAM, V.; BAILEY, D. *Quantification of Brain Function Using PET*. [S.l.]: Academic Press, 1996. Citado na página 17.

NUTTALL, A. H. *Multivariate linear predictive spectral analysis employing weighted forward and backward averaging: A generalization of Burg's algorithm*. [S.l.], 1976. 00049. Disponível em: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA031755>. Citado na página 23.

PERCIVAL, D. B.; WALDEN, A. T. *Spectral Analysis for Physical Applications*. [S.l.]: Cambridge University Press, 1993. Citado na página 25.

PHELPS, M. E. *PET: Physics, Instrumentation, and Scanners*. [S.l.]: Springer, 2006. Citado na página 17.

POLDRACK, R. A.; MUMFORD, J. A.; NICHOLS, T. E. *Handbook of Functional MRI Data Analysis*. [S.l.]: Cambridge University Press, 2011. Citado na página 17.

RAMPIL, M. S.; IRA, J. A primer for eeg signal processing in anesthesia. *Anesthesiology*, v. 89, n. 4, p. 980–1002, 1998. Citado na página 73.

RODRIGUES, P. L.; BACCALA, L. A. A new algorithm for neural connectivity estimation of eeg event related potentials. In: *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*. [S.l.: s.n.], 2015. p. 3787–3790. Citado na página 23.

SAMESHIMA, K.; BACCALá, L. A. (Ed.). *Methods in Brain Connectivity Inference through Multivariate Time Series Analysis*. [S.l.]: CRC Press, Boca Raton, 2014. Citado 2 vezes nas páginas 17 e 25.

SAMESHIMA, K.; BACCALA, L. A. (Ed.). *Methods in Brain Connectivity Inference through Multivariate Time Series Analysis*. Boca Raton: CRC Press, 2014. 245–251 p. Citado na página 22.

STIPPICH, C.; SARTOR, K. *Clinical Functional MRI: Presurgical Functional Neuroimaging*. [S.l.]: Springer, 2007. Citado na página 17.

STROMAN, P. W. *Essentials of Functional MRI*. [S.l.]: Taylor & Francis, 2011. Citado na página 17.

TADEUSZ, B. Zur berechnung der determinanten, wie auchder inversen, und zur darauf basierten auflösung der systeme linearer. *Acta Astronomica*, Series C, n. 1, p. 41—67, 1937. Citado na página 62.

TAKAHASHI, D.; BACCALA, L. A.; SAMESHIMA, K. Information theoretic interpretation of frequency domain connectivity measures. *Biological Cybernetics*, Springer Berlin / Heidelberg, v. 103, p. 463–469, 2010. ISSN 0340-1200. 10.1007/s00422-010-0410-x. Disponível em: <http://dx.doi.org/10.1007/s00422-010-0410-x>. Citado na página 22.

TAKAHASHI, D. Y.; A., B. L.; SAMESHIMA, K. Connectivity inference between neural structures via partial directed coherence. *J. Appl. Stat.*, v. 34, n. 10, p. 1259–1273, 2007. Citado na página 22.

TANENBAUM, A. S. *Redes de computadores*. 5. ed. Amsterdam, Holanda: Pearson Education, 2011. Citado na página 35.

TONG, S.; THAKOR, N. V. *Quantitative EEG Analysis Methods and Applications*. [S.l.]: Artech House, 2009. Citado na página 17.

VARSAVSKY, A.; MAREELS, I.; COOK, M. *Epileptic Seizures and the EEG: Measurement, Models, Detection and Prediction*. [S.l.]: CRC Press, 2010. Citado na página 18.

WHITTLE, P. On fitting of multivariable autoregressions and the approximate canonical factorization of a spectral density matrix. *Biometrika*, v. 50, p. 129–134, 1963. Citado na página 23.

Anexos

ANEXO A – Resultados do PDC (u_{5x2000}, ordem=6)

Os gráficos apresentados nesta sessão são referentes ao resultado da execução completa do cálculo de PDC, que resultou em uma matriz tridimensional R_{ijk} e serão mostrados os vetores na posição i,j para k={1:64}.

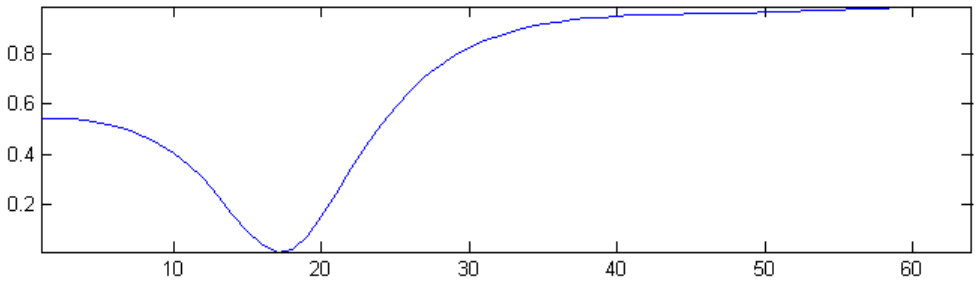

Figura 39: R_{11}

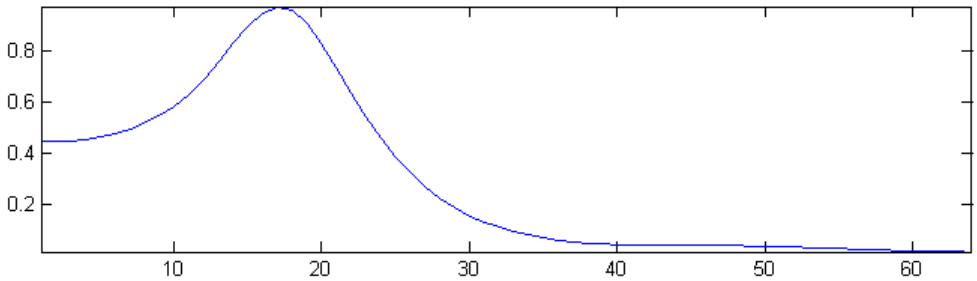

Figura 40: R_{21}

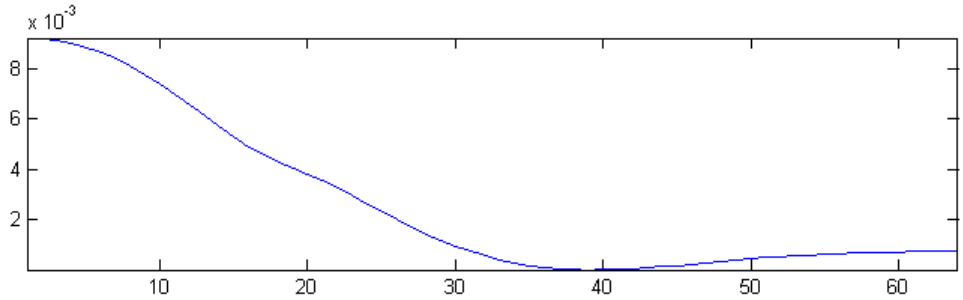

Figura 41: R_{31}

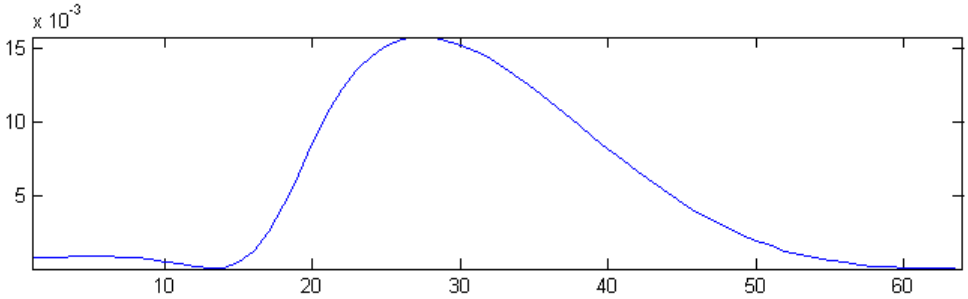

Figura 42: R_{41}

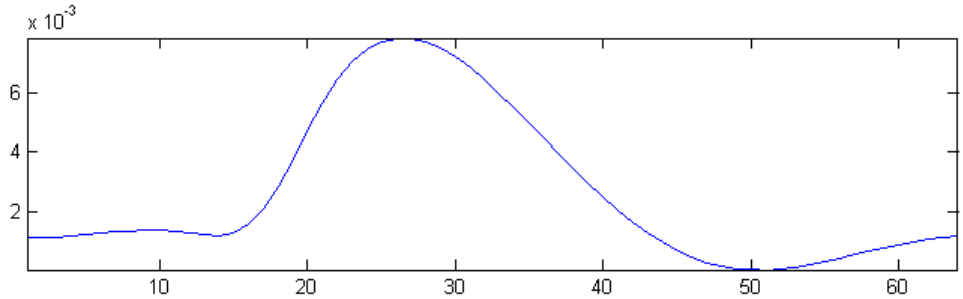

Figura 43: R_{51}

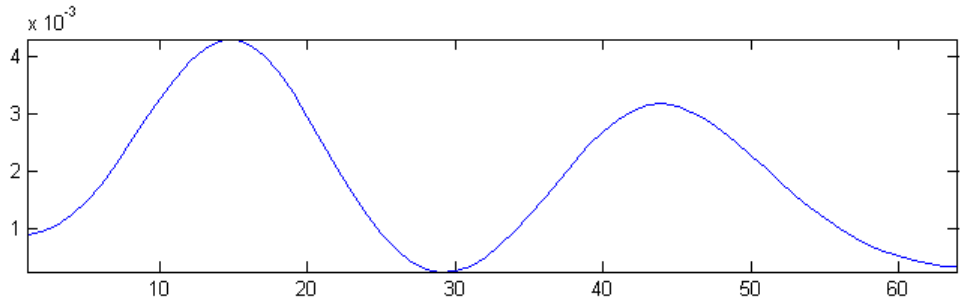

Figura 44: R_{12}

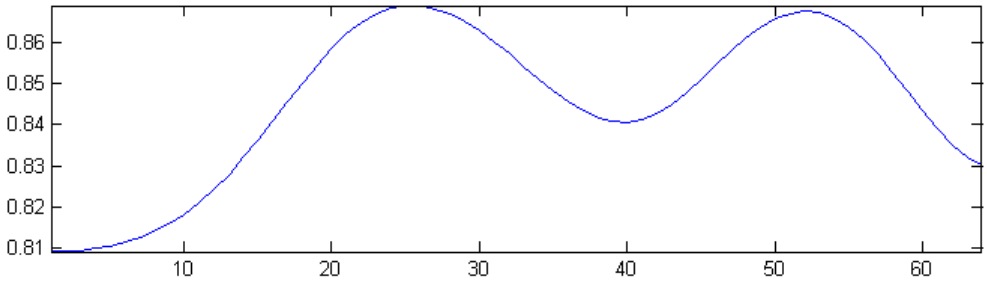

Figura 45: R_{22}

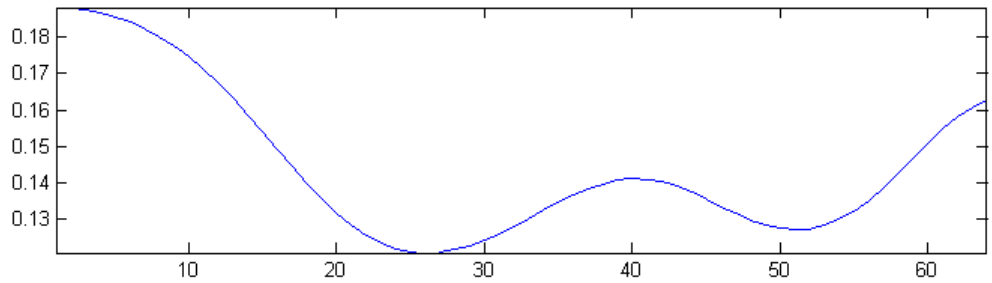

Figura 46: R_{32}

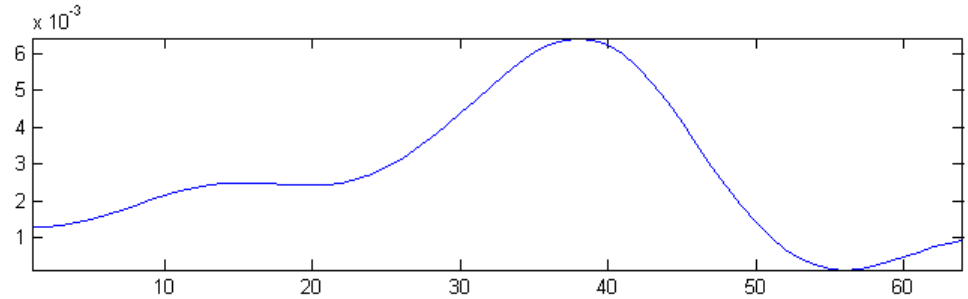

Figura 47: R_{42}

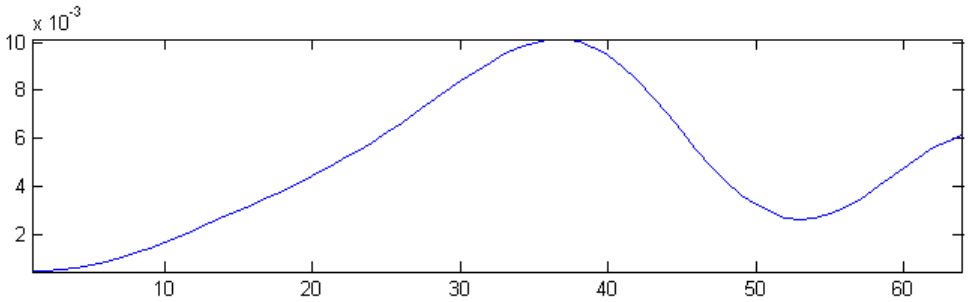

Figura 48: R_{52}

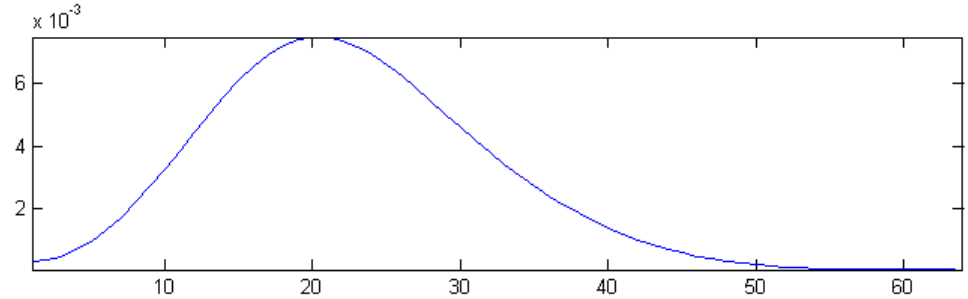

Figura 49: R_{13}

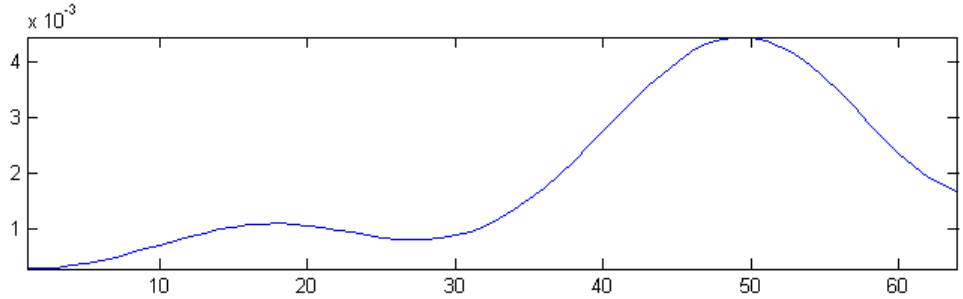

Figura 50: R_{23}

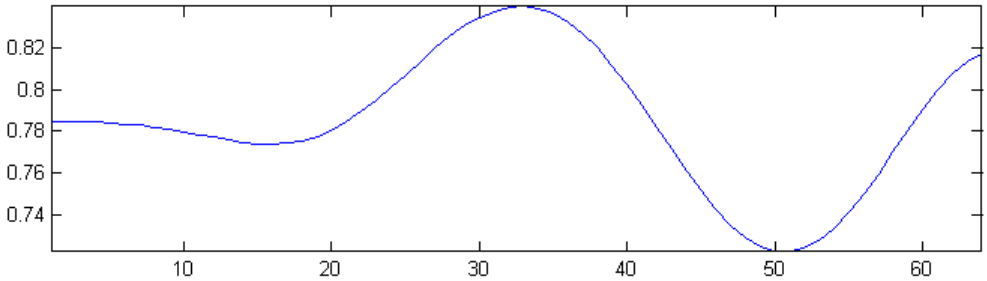

Figura 51: R_{33}

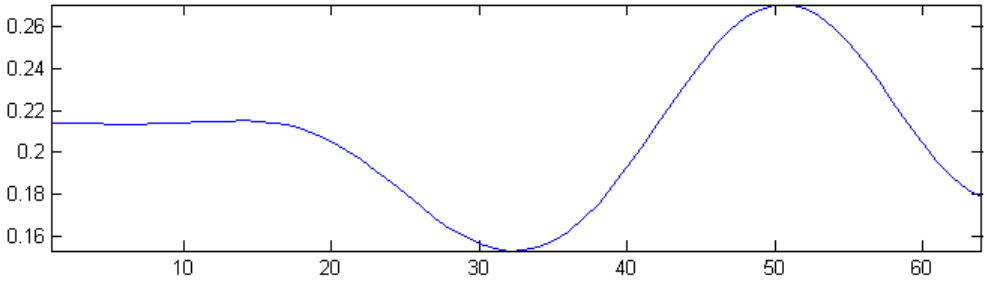

Figura 52: R_{43}

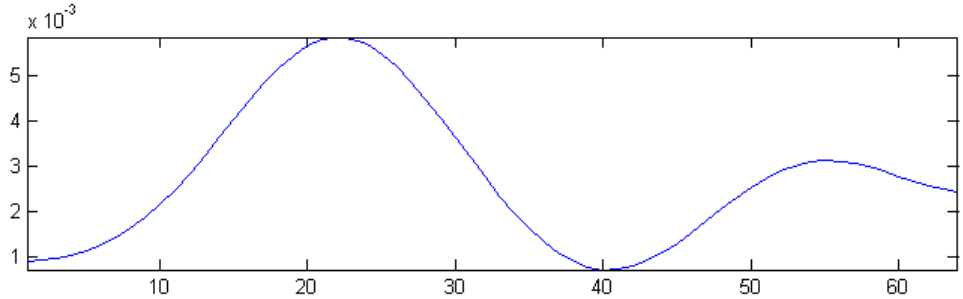

Figura 53: R_{53}

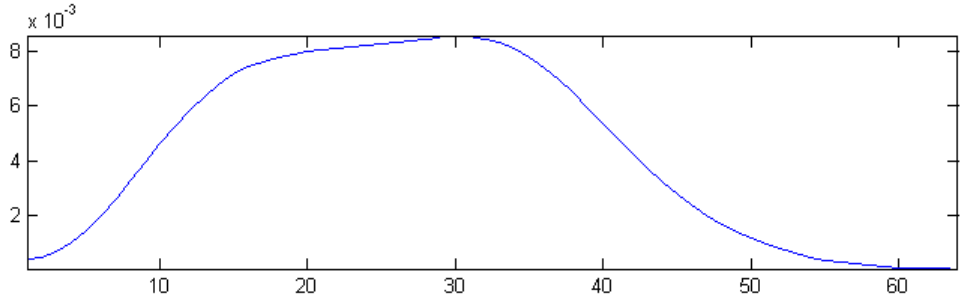

Figura 54: R_{14}

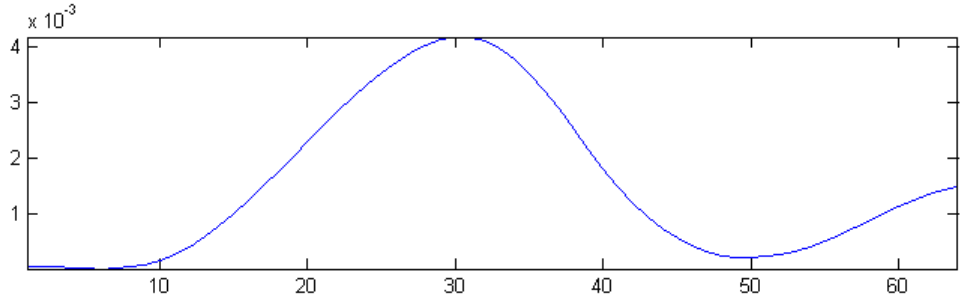

Figura 55: R_{24}

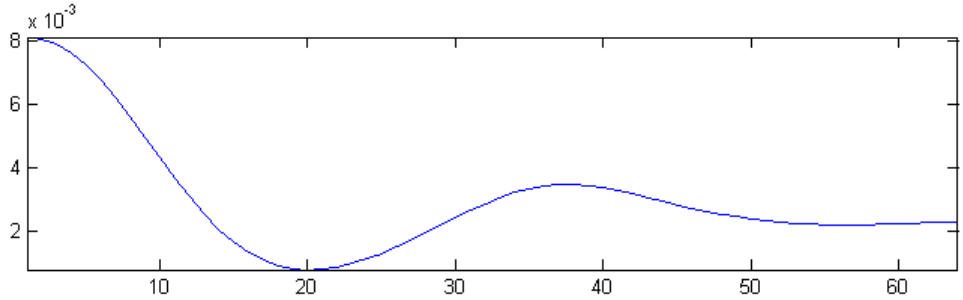

Figura 56: R_{34}

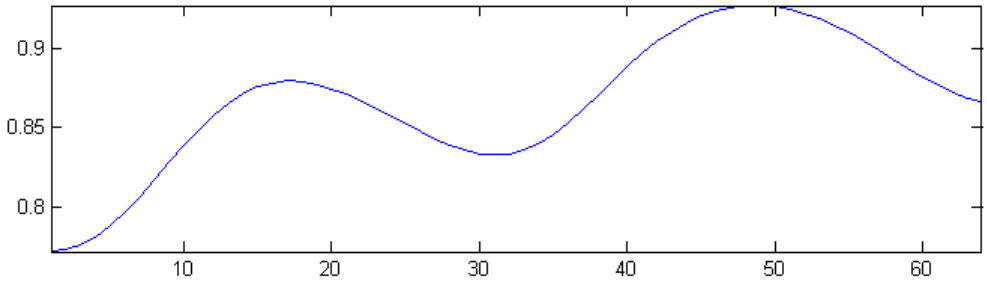

Figura 57: R_{44}

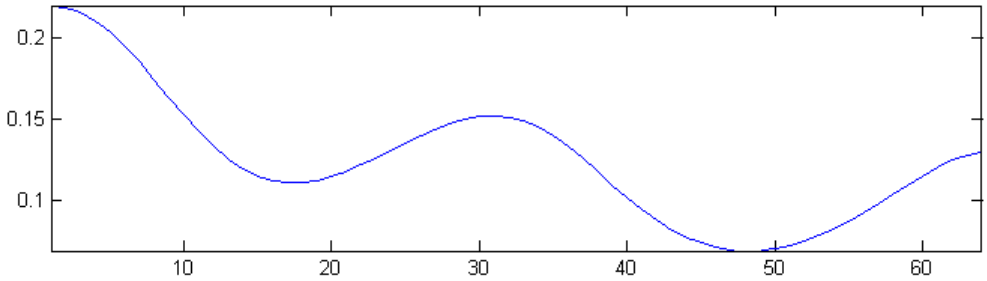

Figura 58: R_{54}

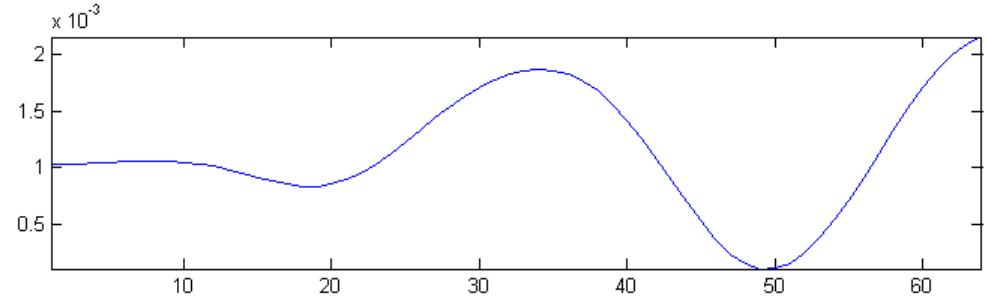

Figura 59: R_{15}

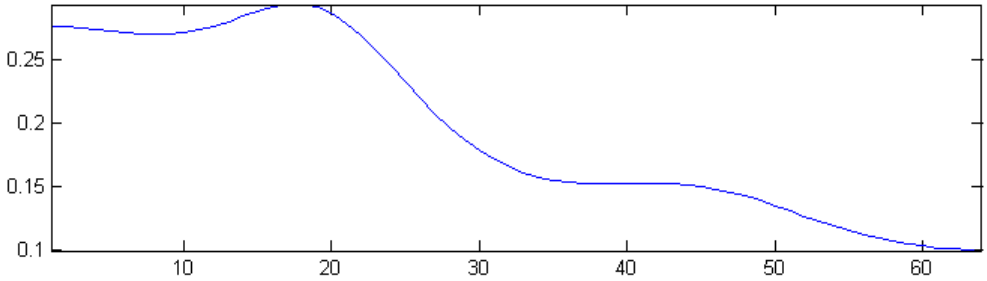

Figura 60: R_{25}

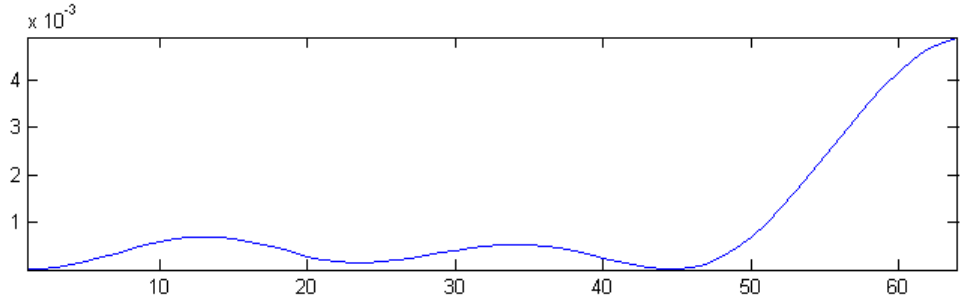

Figura 61: R_{35}

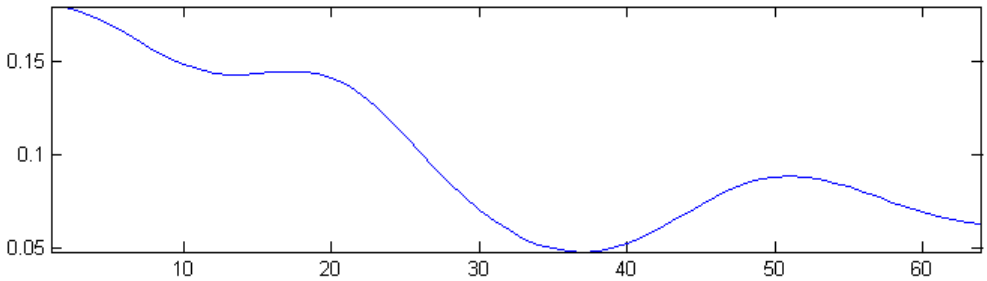

Figura 62: R_{45}

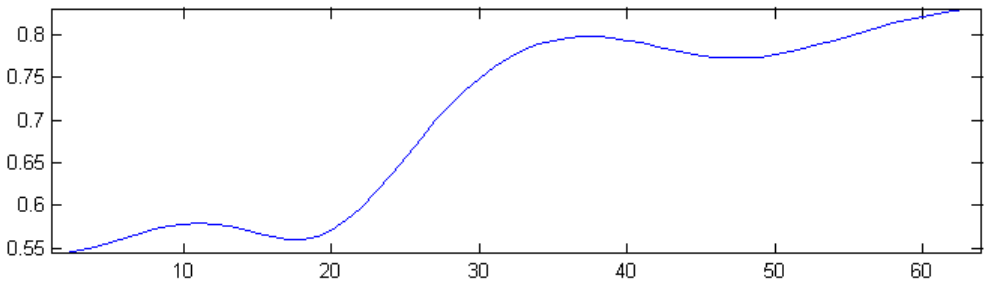

Figura 63: R_{55}

ANEXO B – Teste de desempenho resultados

Arquivo gerado automaticamente durante o teste de desempenho, a unidade de tempo é milissegundos.

Teste de desempenho

(Repeticoes por teste = 10)

(N = 400) (canais = 2) (Ordem = 3) Mcarns: (3.8487) (3.6130) (1.4440) (1.4310) (1.3977) (1.3837) (1.4137) (1.3707) (1.3753) (1.3780) PDC: (6.0490) (2.5097) (2.2283) (2.2077) (2.2060) (2.2080) (2.2117) (2.2150) (2.2480) (2.2133) MEDIA: (Mcarns = 1.8656) (PDC = 2.6297)

(N = 400) (canais = 2) (Ordem = 6) Mcarns: (5.0097) (2.1763) (2.1227) (3.6317) (2.2040) (2.1587) (2.1703) (2.1903) (2.1480) (2.2023) PDC: (2.2520) (2.2180) (2.3530) (2.2407) (2.2210) (2.2117) (2.2403) (2.2193) (2.2340) (2.2180) MEDIA: (Mcarns = 2.6014) (PDC = 2.2408)

(N = 400) (canais = 2) (Ordem = 9) Mcarns: (5.4637) (2.9840) (3.3273) (3.3093) (3.0200) (3.0020) (3.0287) (2.9887) (2.9783) (2.9957) PDC: (2.2310) (2.2183) (2.6267) (2.3553) (2.2240) (2.2460) (2.2250) (2.2313) (2.2250) (2.2290) MEDIA: (Mcarns = 3.3098) (PDC = 2.2812)

(N = 400) (canais = 2) (Ordem = 12) Mcarns: (7.3977) (3.6600) (5.3847) (3.8497) (3.7600) (3.6960) (3.7603) (3.7180) (3.7690) (3.7057) PDC: (2.2390) (2.3630) (2.2457) (2.2260) (2.2967) (2.2200) (2.2497) (2.2230) (2.2253) (2.2383) MEDIA: (Mcarns = 4.2701) (PDC = 2.2527)

(N = 400) (canais = 2) (Ordem = 15) Mcarns: (9.6970) (5.3573) (4.5780) (4.5353) (4.5267) (4.4740) (4.5980) (4.5453) (4.4790) (4.5223) PDC: (2.2537) (2.7383) (2.2470) (2.2207) (2.2230) (2.2340) (2.2213) (2.2190) (2.2247) (2.2283) MEDIA: (Mcarns = 5.1313) (PDC = 2.2810)

(N = 400) (canais = 2) (Ordem = 18) Mcarns: (6.7483) (5.4117) (6.4990) (5.3110) (5.2890) (5.3023) (5.2860) (5.2643) (5.3450) (5.2733) PDC: (2.2653) (2.5007) (2.2587) (2.2550) (2.2230) (2.2273) (2.2287) (2.2527) (2.2237) (2.2307) MEDIA: (Mcarns = 5.5730) (PDC = 2.2666)

(N = 400) (canais = 2) (Ordem = 21) Mcarns: (7.1827) (6.3940) (6.5543)

(6.1403) (6.1070) (6.1827) (6.1223) (6.1403) (6.1590) (6.1407) PDC: (2.2413) (2.6153) (2.2560) (2.2357) (2.2280) (2.2563) (2.2247) (2.2280) (2.2270) (2.2293) MEDIA: (Mcarns = 6.3123) (PDC = 2.2742)

(N = 400) (canais = 2) (Ordem = 24) Mcarns: (9.0700) (8.8880) (6.9057) (6.9330) (6.8920) (6.9267) (6.8900) (6.9557) (6.9340) (6.9120) PDC: (2.2587) (2.2647) (2.2310) (2.2317) (2.2367) (2.2307) (2.2290) (2.2277) (2.2297) (2.2347) MEDIA: (Mcarns = 7.3307) (PDC = 2.2374)

(N = 400) (canais = 2) (Ordem = 27) Mcarns: (11.2710) (9.8330) (7.6707) (7.6577) (7.7143) (7.7420) (7.6963) (7.7147) (7.7370) (7.7683) PDC: (2.2620) (2.2967) (2.2557) (2.2587) (2.2580) (2.2547) (2.2507) (2.2540) (2.2557) (2.2573) MEDIA: (Mcarns = 8.2805) (PDC = 2.2603)

(N = 400) (canais = 2) (Ordem = 30) Mcarns: (13.4537) (10.4197) (8.5293) (8.5263) (8.4673) (8.5000) (8.5687) (8.5520) (8.5597) (8.5920) PDC: (2.2903) (2.3020) (2.2593) (2.2330) (2.2290) (2.2330) (2.2297) (2.2323) (2.2380) (2.2353) MEDIA: (Mcarns = 9.2169) (PDC = 2.2482)

(N = 400) (canais = 3) (Ordem = 3) Mcarns: (2.6667) (2.5397) (2.5730) (2.5317) (2.5110) (2.5530) (2.4913) (2.5090) (2.5127) (2.5397) PDC: (4.9217) (4.9033) (4.9400) (4.9310) (4.8983) (4.9013) (4.9147) (4.9340) (4.9003) (4.9007) MEDIA: (Mcarns = 2.5428) (PDC = 4.9145)

(N = 400) (canais = 3) (Ordem = 6) Mcarns: (4.3990) (4.9843) (4.3747) (4.2243) (4.2960) (4.2880) (4.3090) (4.3200) (4.2897) (4.2830) PDC: (4.9710) (5.3657) (4.9467) (4.9203) (4.9023) (4.9113) (4.9010) (4.9057) (4.9010) (4.9060) MEDIA: (Mcarns = 4.3768) (PDC = 4.9631)

(N = 400) (canais = 3) (Ordem = 9) Mcarns: (10.6203) (6.1177) (6.9713) (6.0610) (6.0937) (6.5847) (6.2380) (6.1717) (6.1693) (6.1220) PDC: (4.9490) (5.5417) (4.9690) (4.9250) (4.9463) (4.9387) (4.9183) (4.9180) (4.9213) (4.9347) MEDIA: (Mcarns = 6.7150) (PDC = 4.9962)

(N = 400) (canais = 3) (Ordem = 12) Mcarns: (17.9497) (7.9470) (7.9903) (7.9220) (7.8963) (7.9193) (7.9953) (7.9447) (7.8777) (7.8837) PDC: (4.9917) (4.9477) (4.9143) (4.9067) (4.9317) (4.9140) (4.9087) (4.9330) (4.9383) (4.9117) MEDIA: (Mcarns = 8.9326) (PDC = 4.9298)

ANEXO C – Gráficos MatLab x RTMcarns todas otimizações

Seguem os gráficos de todas as amostras criadas para o teste de desempenho.

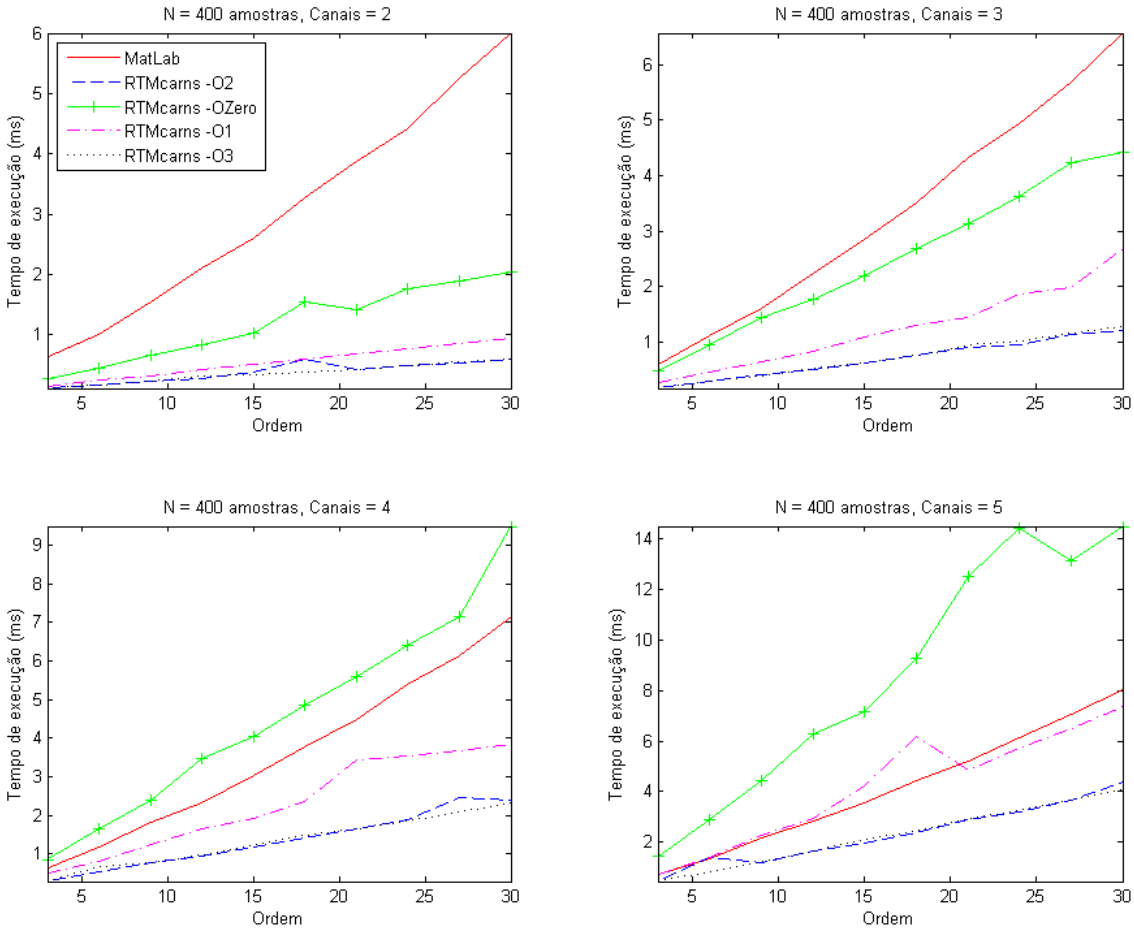

Figura 64: Desempenho MatLab x RTMcarns todas otimizações para u_{2x400}, u_{3x400}, u_{4x400} e u_{5x400}

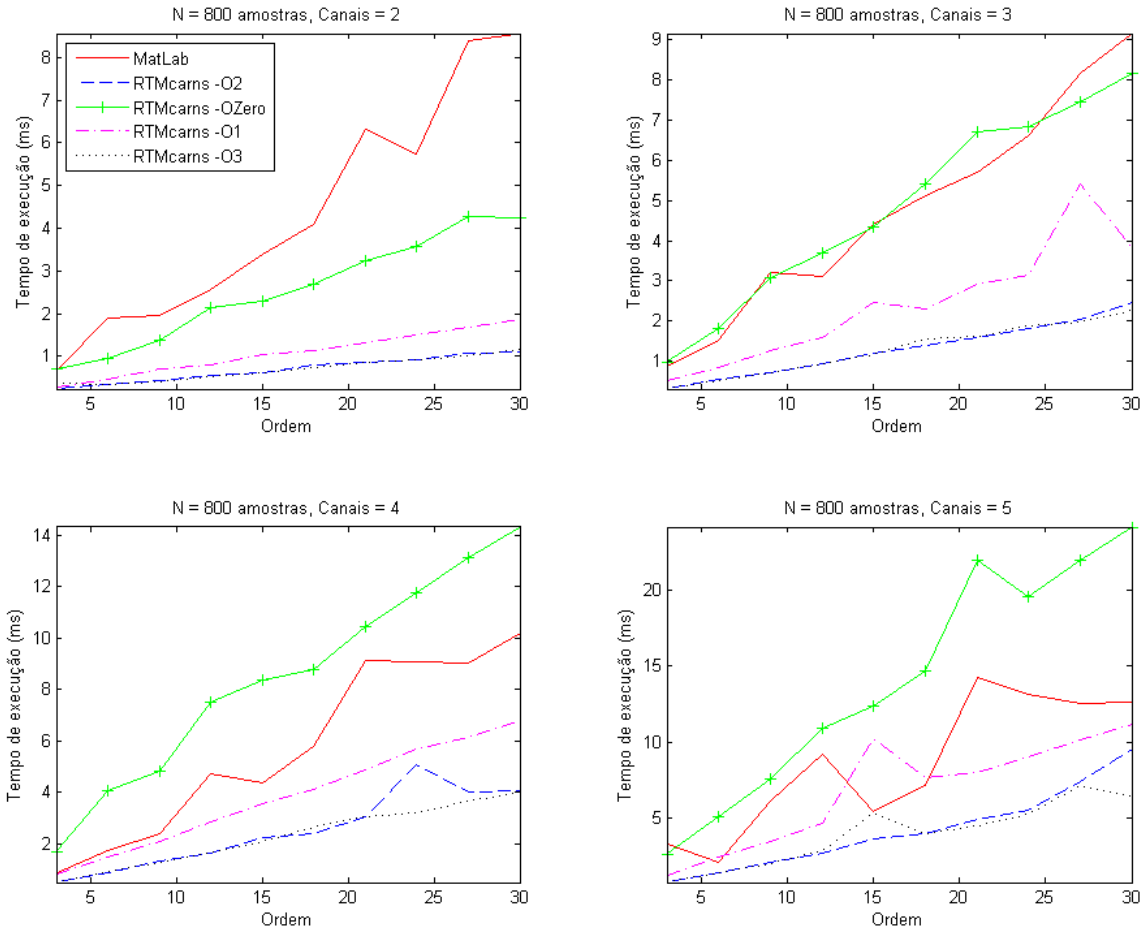

Figura 65: Desempenho MatLab x RTMcarns todas otimizações para u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

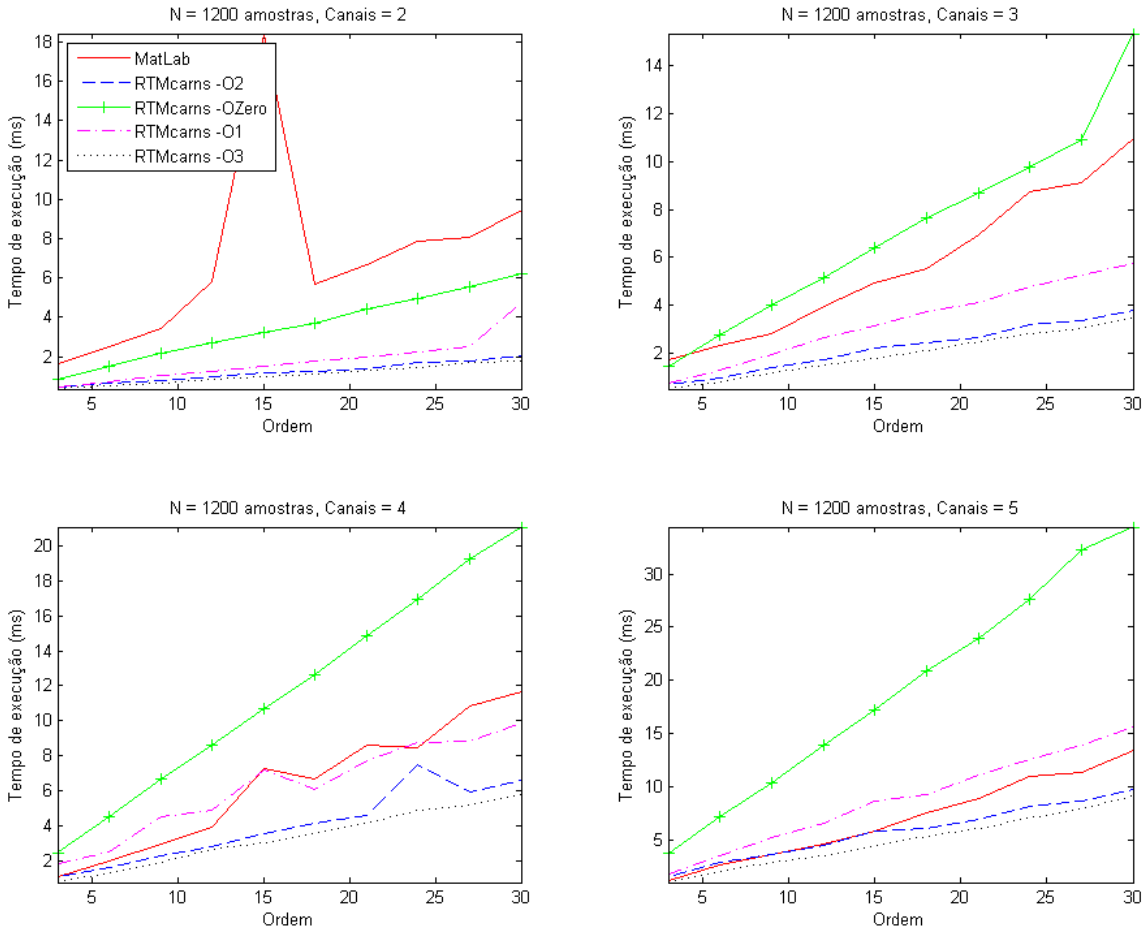

Figura 66: Desempenho MatLab x RTMcarns todas otimizações para u_{2x1200}, u_{3x1200}, u_{4x1200} e u_{5x1200}

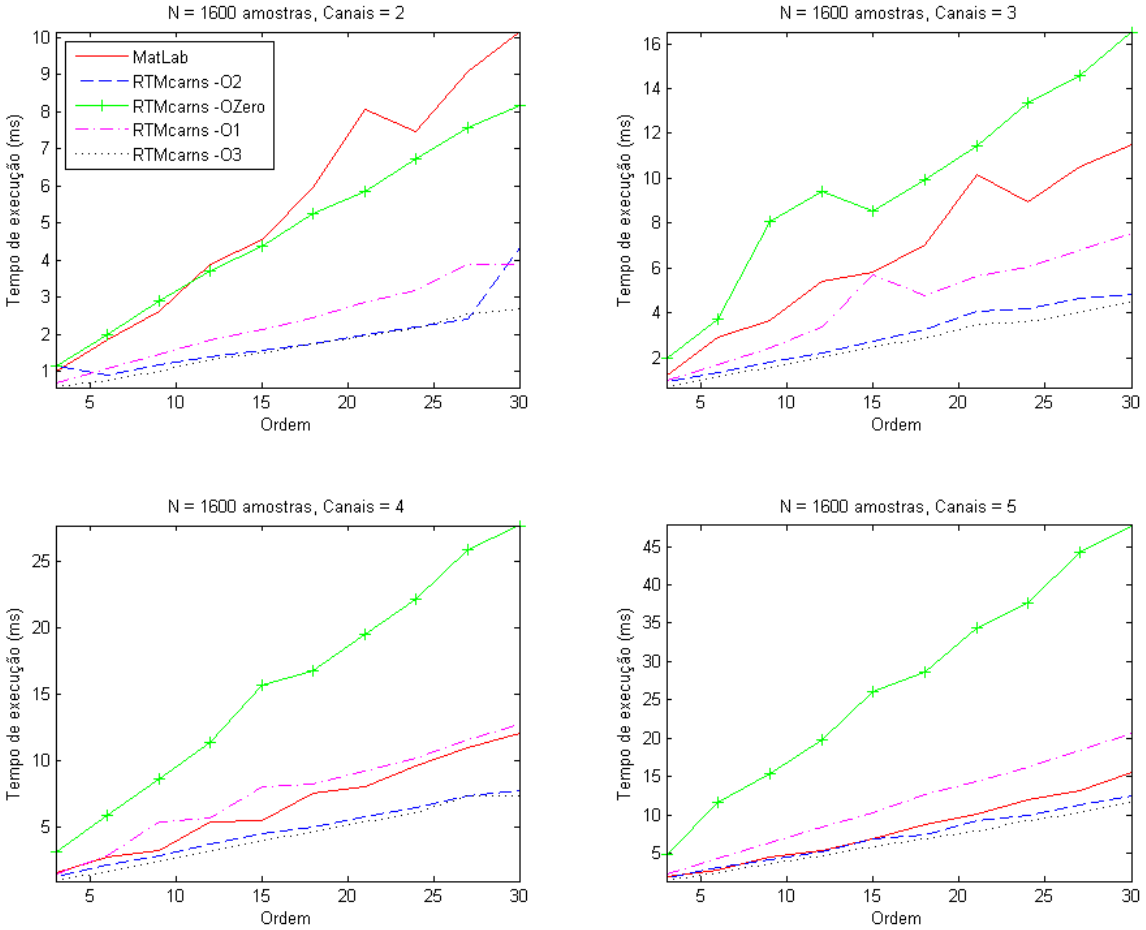

Figura 67: Desempenho MatLab x RTMcarns todas otimizações para u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

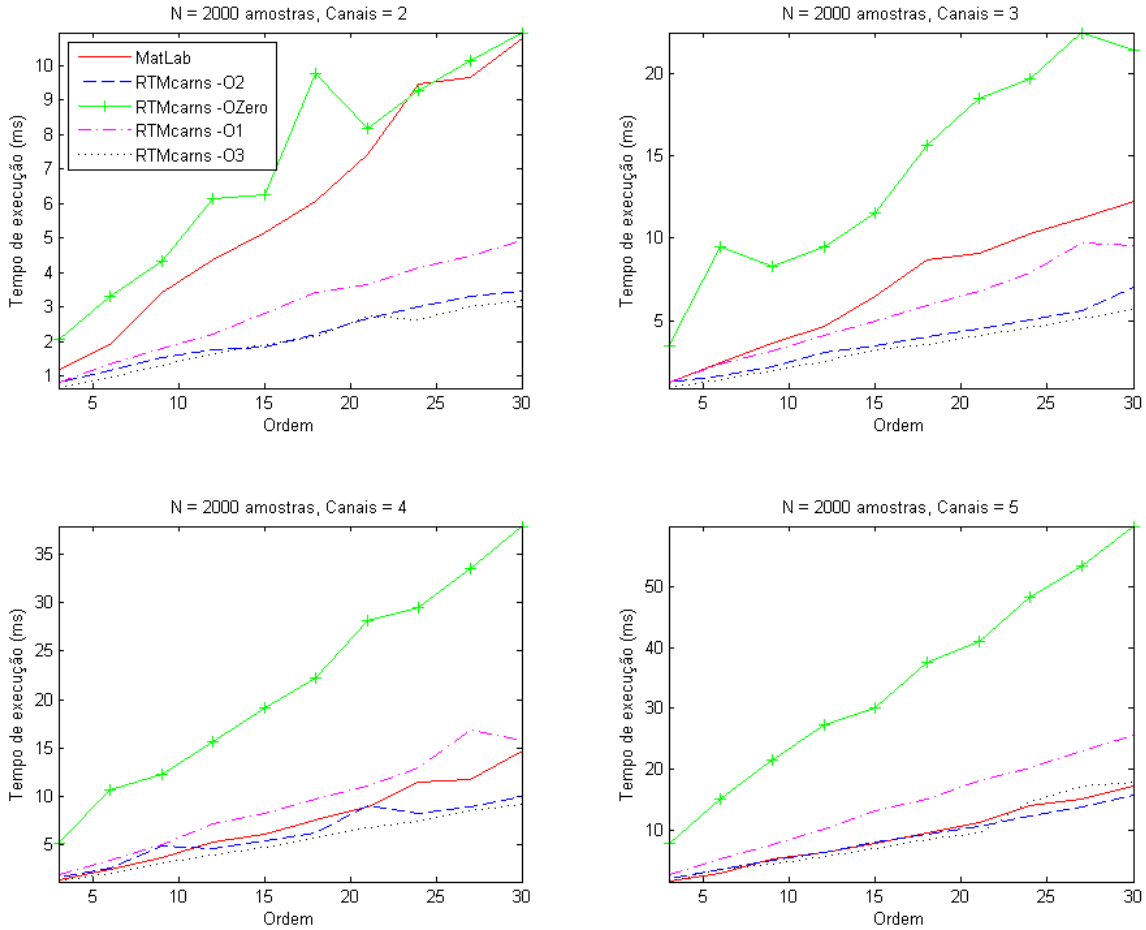

Figura 68: Desempenho MatLab x RTMcarns todas otimizações para u_{2x2000}, u_{3x2000}, $u_{4x82000}$ e u_{5x2000}

ANEXO D – Desempenho da função PDC

Seguem os gráficos do desempenho da função PDC para todos os sinais, comparando MatLab e RTMcanrs -O3.

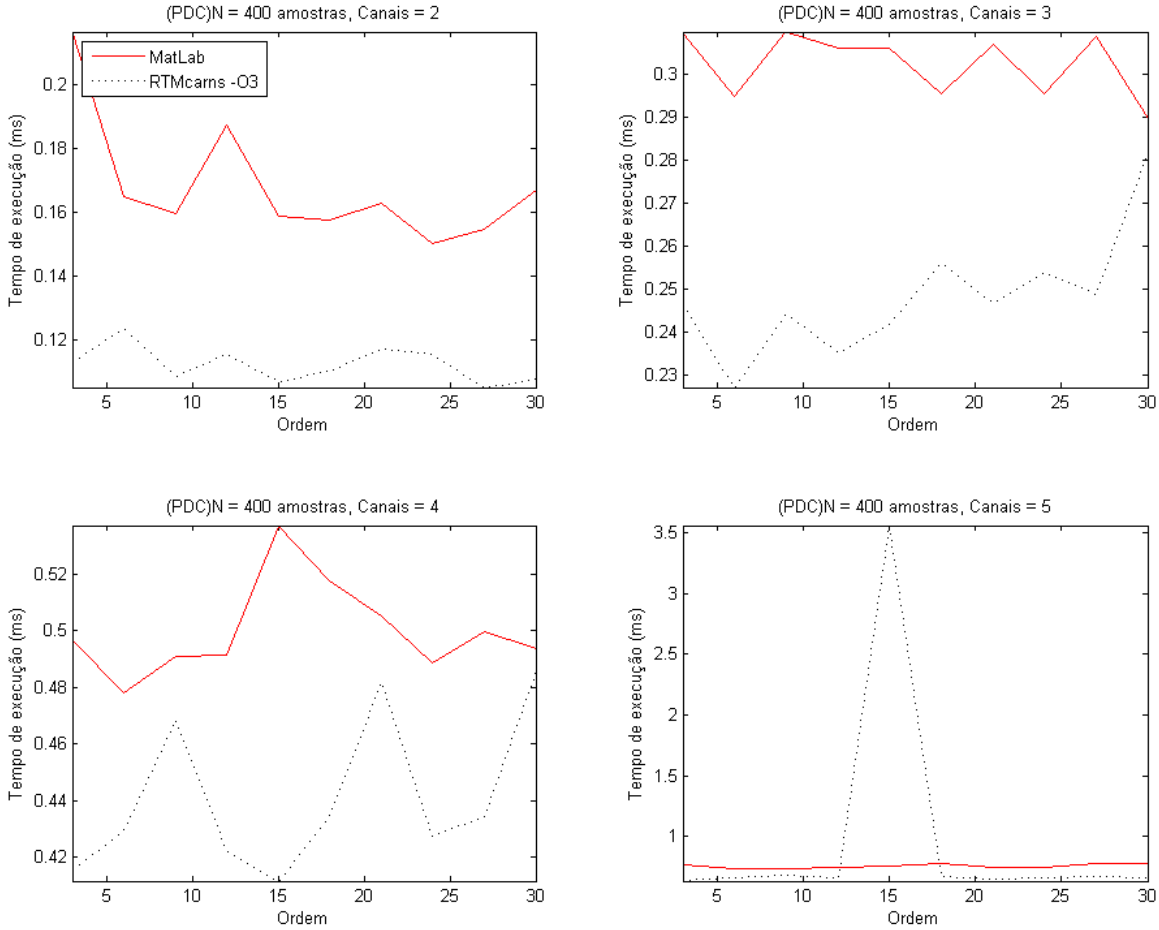

Figura 69: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x400}, u_{3x400}, u_{4x400} e u_{5x400}

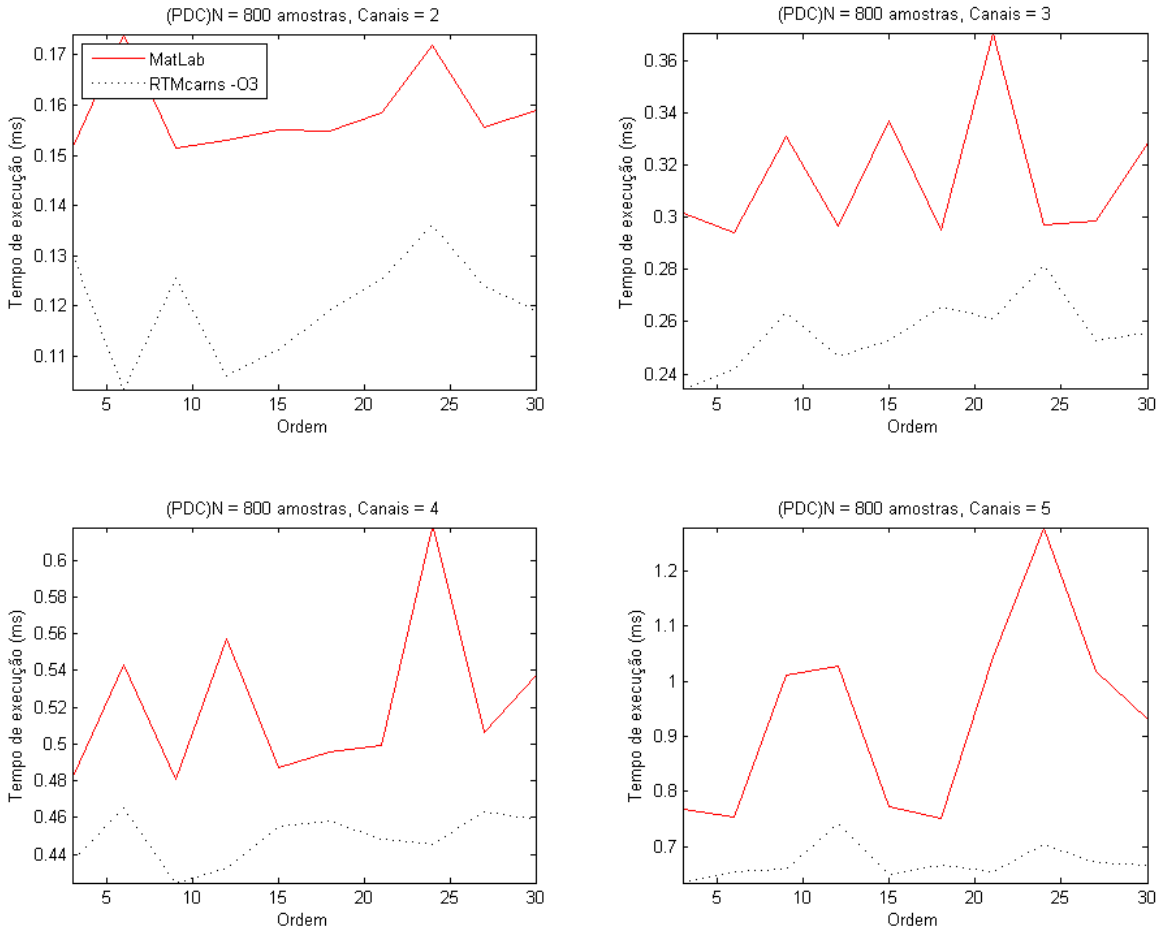

Figura 70: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

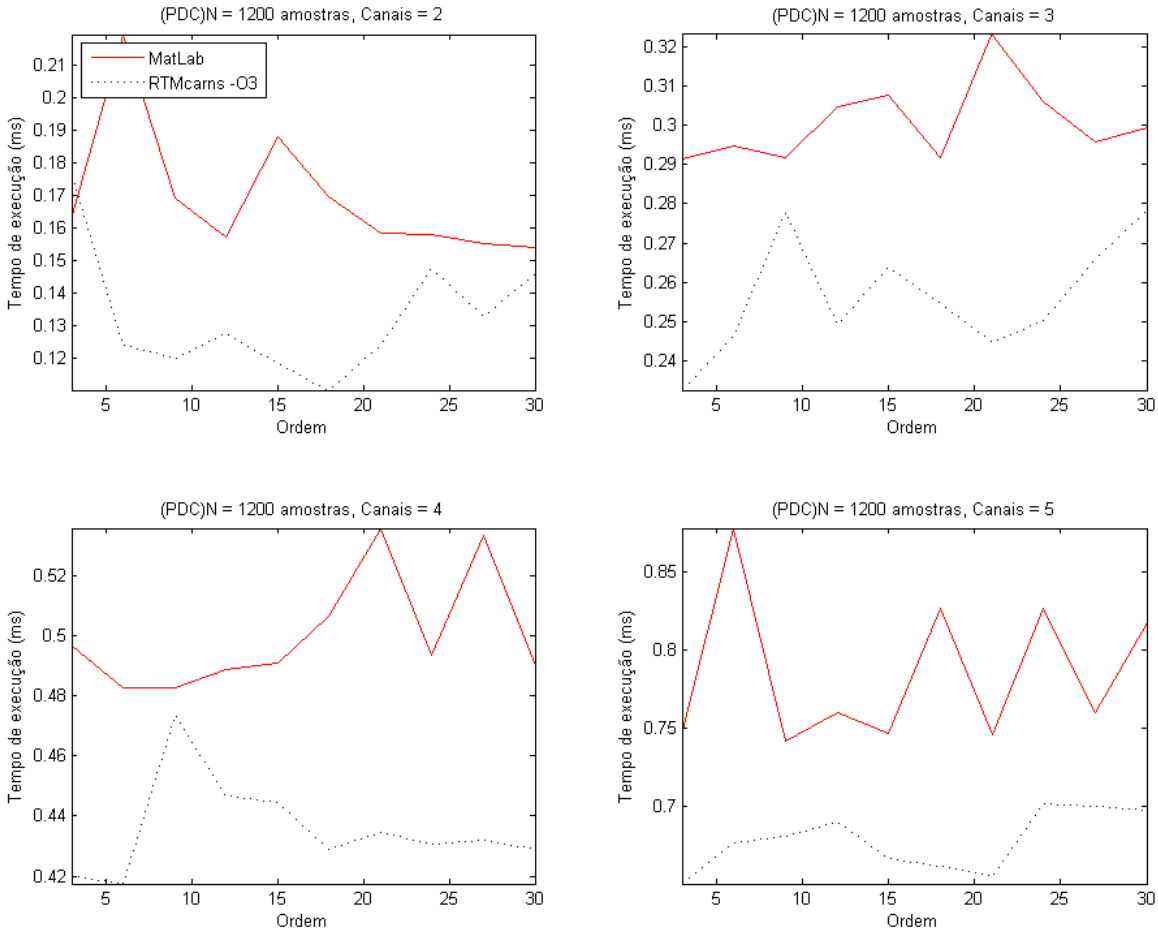

Figura 71: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x1200}, u_{3x1200}, u_{4x1200} e u_{5x1200}

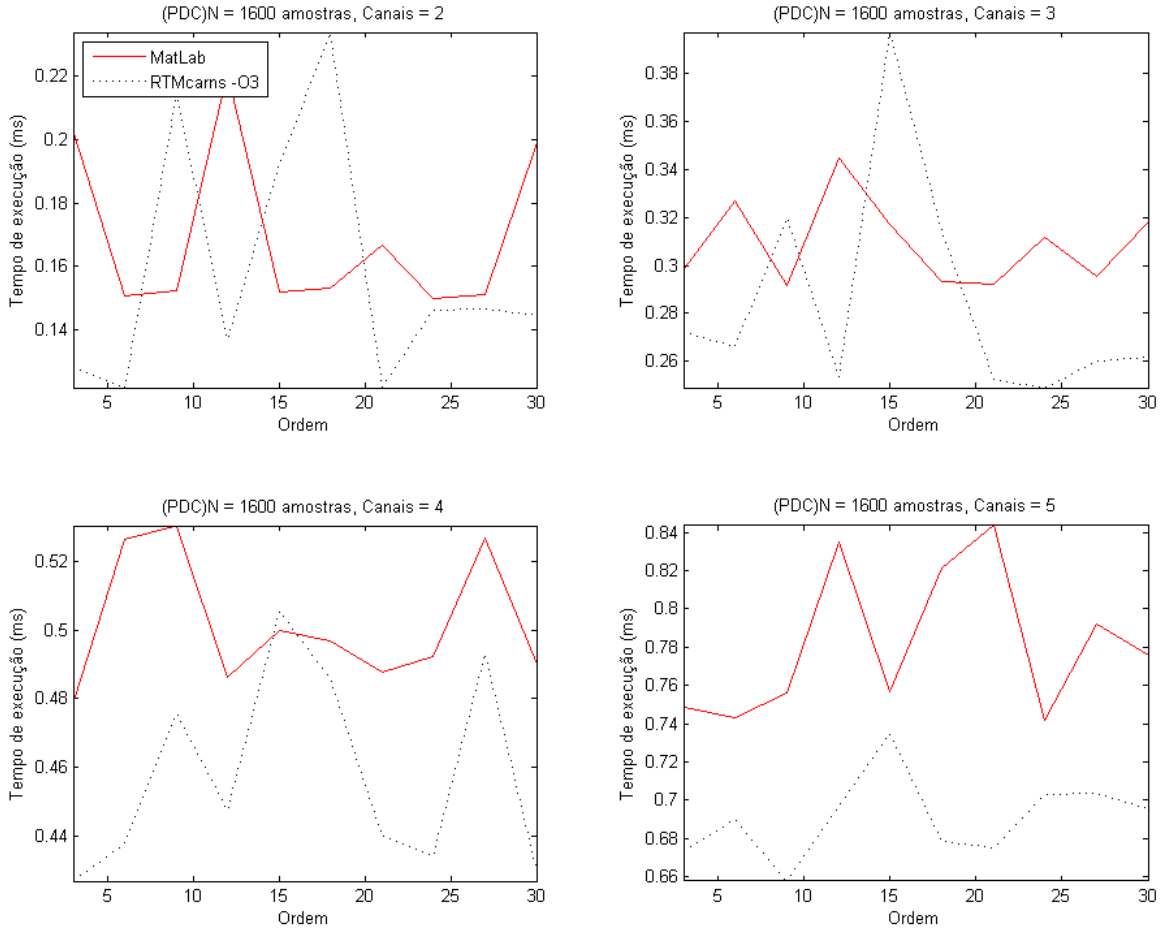

Figura 72: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

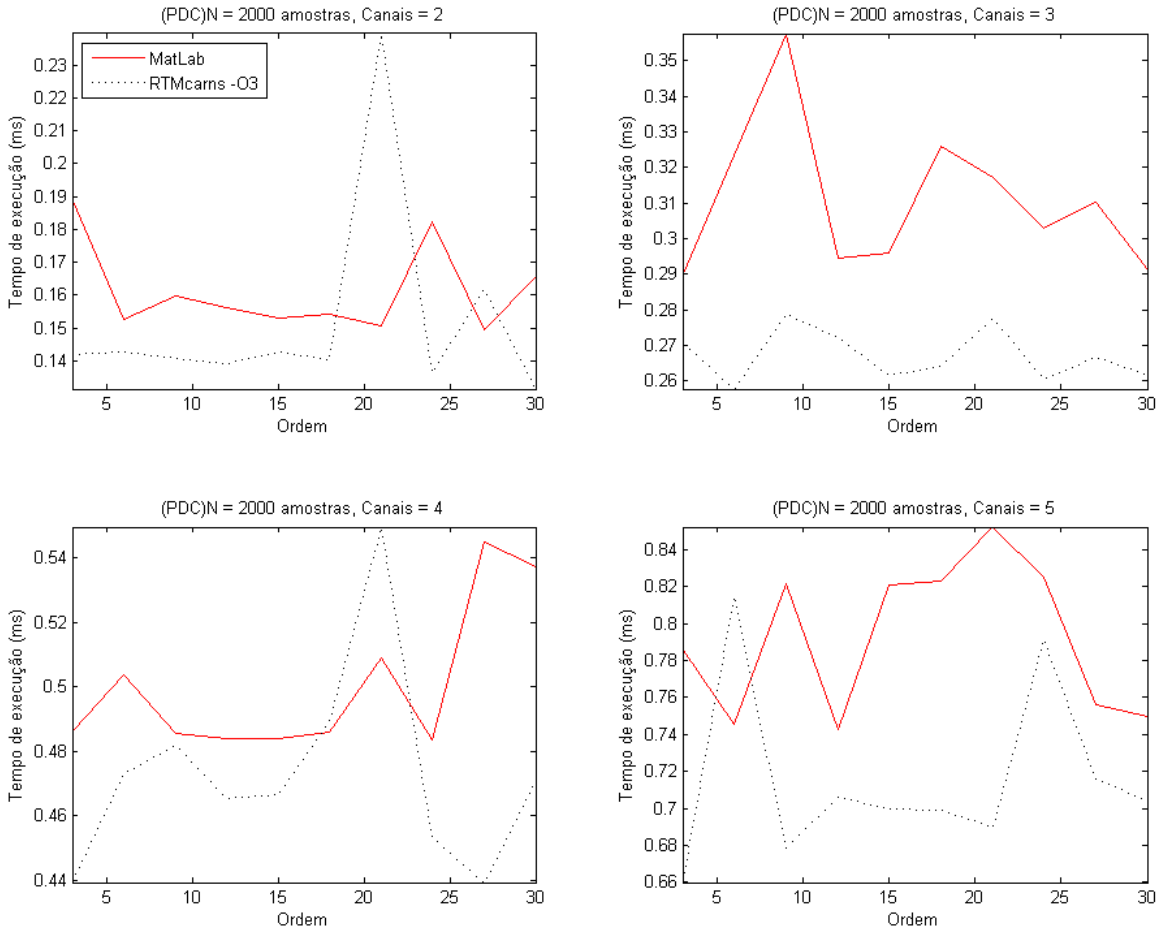

Figura 73: Desempenho da função PDC do MatLab x RTMcarns -O3 para u_{2x2000}, u_{3x2000}, u_{4x2000} e u_{5x2000}

ANEXO E – Desempenho do Mcarns no embarcado

Segue os gráficos de todos os sinais.

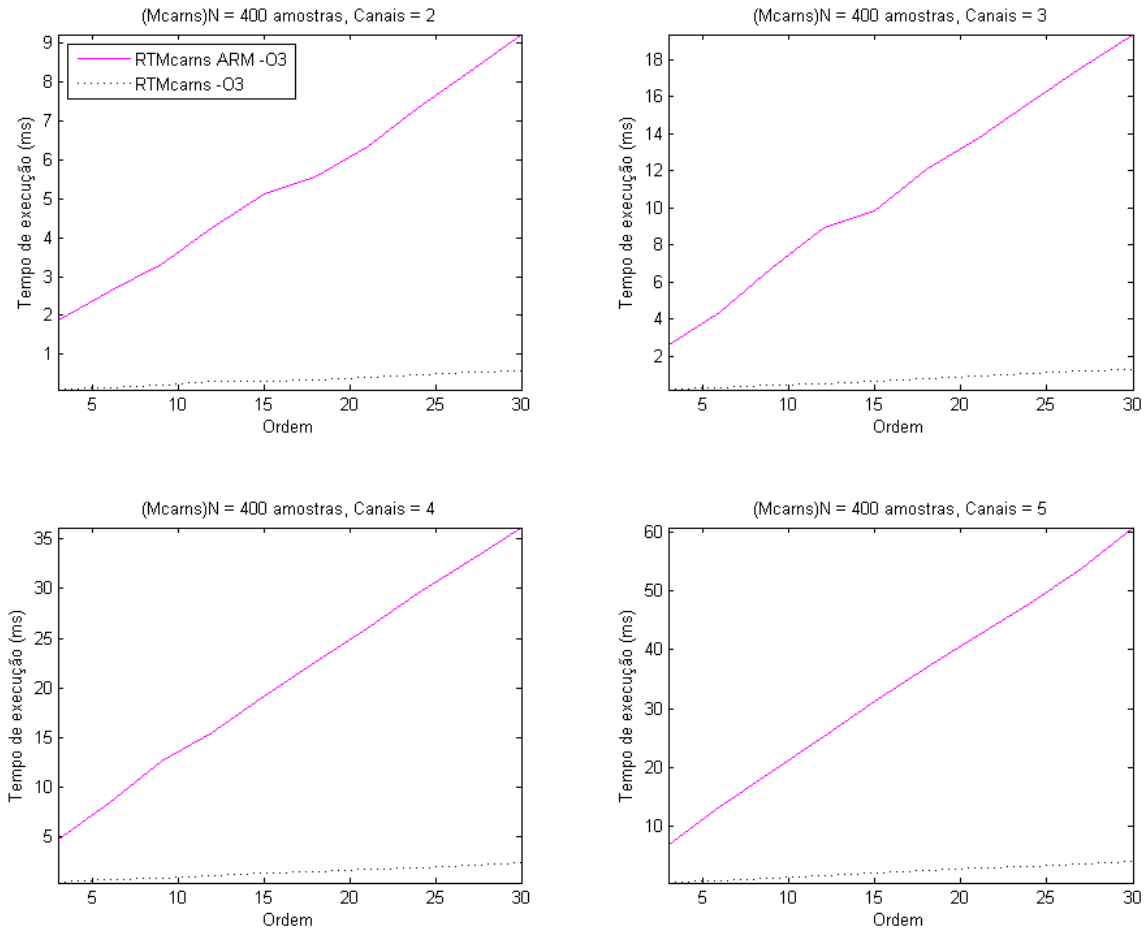

Figura 74: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x400}, u_{3x400}, u_{4x400} e u_{5x400}

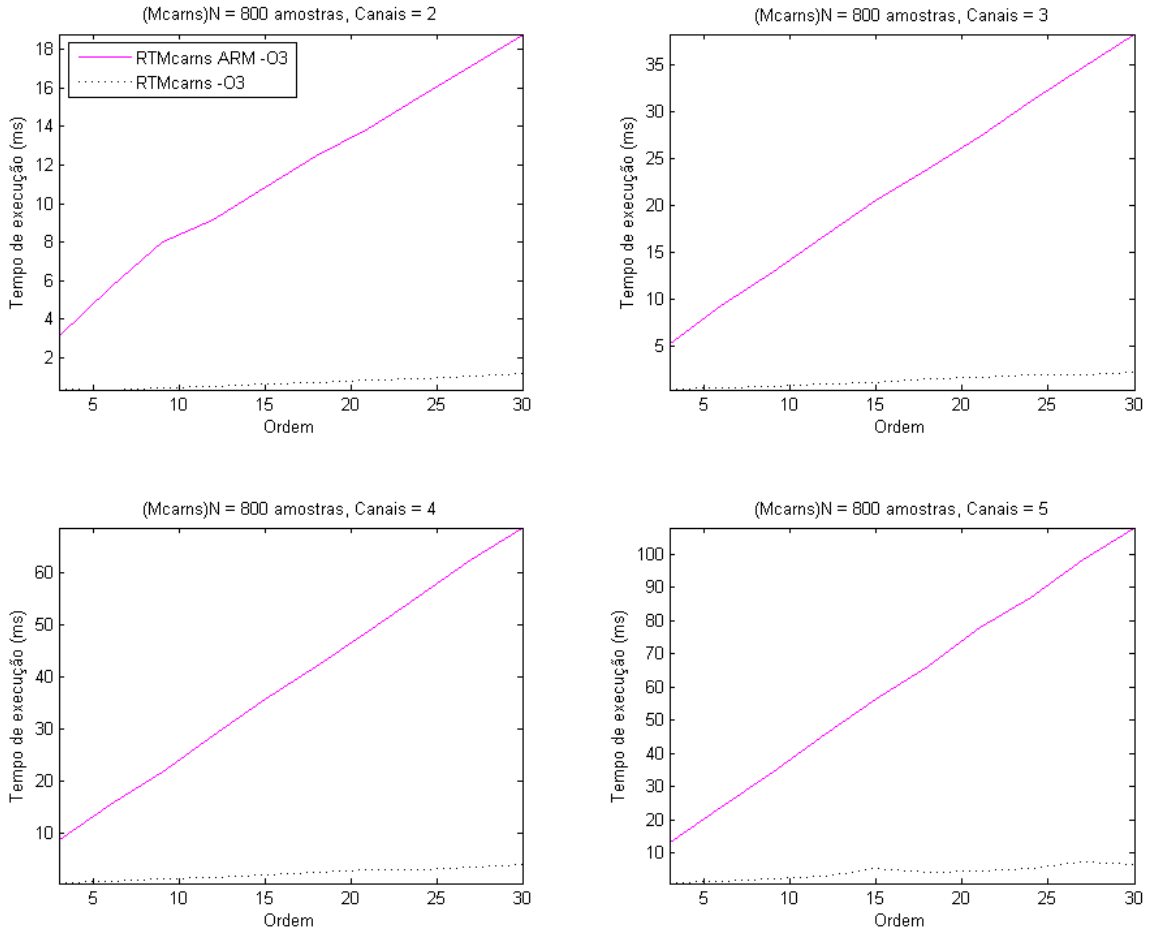

Figura 75: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x800}, u_{3x800}, u_{4x800} e u_{5x800}

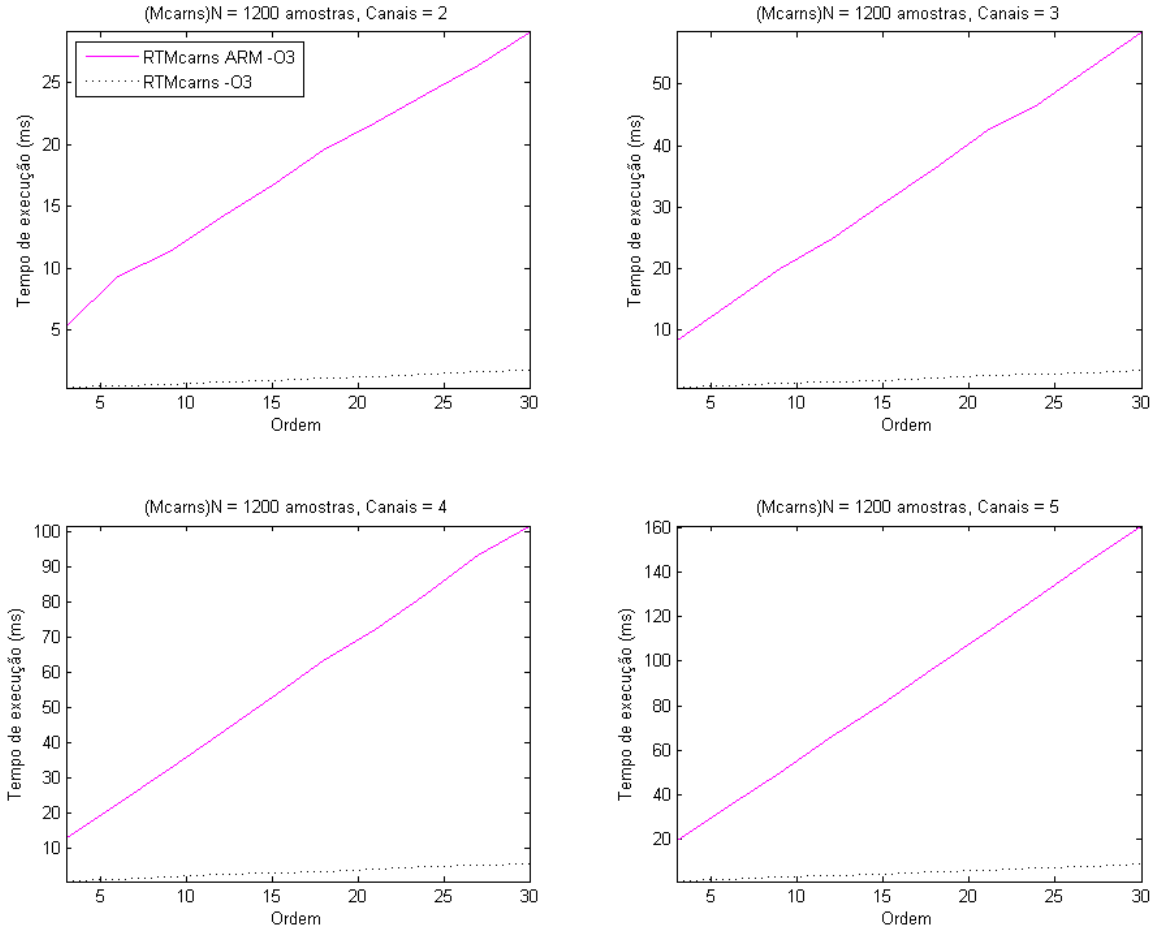

Figura 76: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x1200}, u_{3x1200}, u_{4x1200} e u_{5x1200}

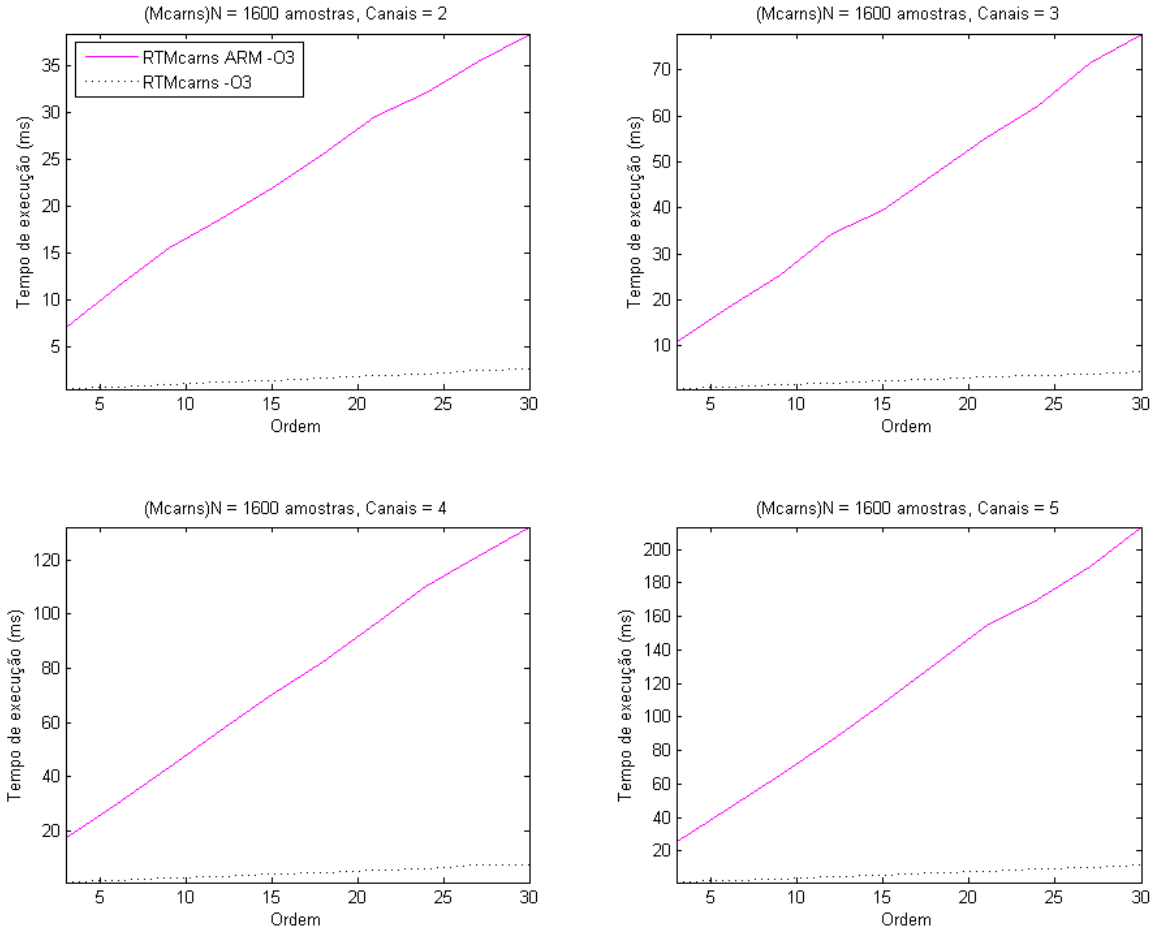

Figura 77: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x1600}, u_{3x1600}, u_{4x1600} e u_{5x1600}

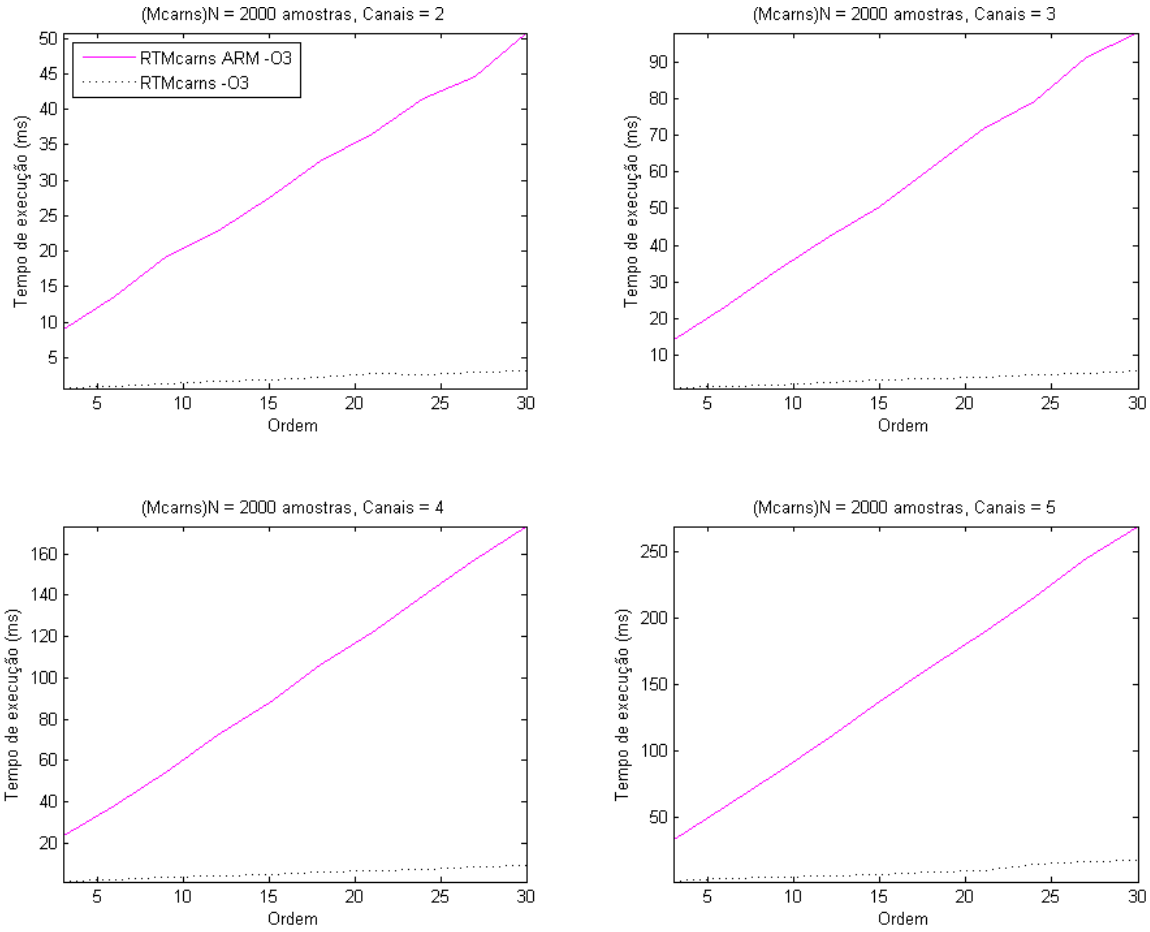

Figura 78: Desempenho da função Mcarns do ARM x computador para os sinais u_{2x2000}, u_{3x2000}, u_{4x2000} e u_{5x2000}