Universidade do Algarve

Faculdade de Ciências e Tecnologia

Análise Diferenciada das Componentes Independentes de Potenciais Evocados P3a e P3b

Mestrado em Imagiologia Médica

Michael Guerreiro

FARO

2007

Título da Dissertação:	Análise Diferenciada das Componentes Independentes de Potenciais Evocados P3a e P3b
Nome:	Michael Guerreiro
Orientador:	Doutora Maria da Graça Cristo dos Santos Lopes Ruano
Co-Orientador:	Doutora Carla Maria Quintão Pereira da Silva
Data:	2 de Maio de 2007
Júri:	Doutora Maria da Conceição Abreu e Silva Doutora Maria da Graça Cristo dos Santos Lopes Ruano Doutor João Paulo Trigueiros da Silva Cunha Doutora Carla Maria Quintão Pereira da Silva

O conteúdo deste relatório é da exclusiva reponsabilidade do autor

Agradecimentos

Não poderia deixar de aqui dedicar uma palavra de agradecimento às pessoas que, em vários aspectos, me auxiliaram na elaboração deste trabalho. Sem a sua preciosa colaboração, certamente encontraria maiores dificuldades em encontrar bibliografia, ferramentas para o desenvolvimento do trabalho e respostas para as dúvidas que fui tendo ao longo deste trabalho.

Agradeço à Prof[a]. Doutora Carla Maria Silva pelo auxílio, pelo esclarecimento das minhas dúvidas e pela disponibilidade em facilitar a bibliografia necessária. A Doutora Carla esteve sempre disponível para ajudar e aconselhar no que fosse necessário.

Agradeço à Prof[a]. Doutora Maria da Graça Ruano pelo auxílio, pelo esclarecimento das minhas dúvidas, por toda a dedicação despendida e por me ter convencido a entrar neste Mestrado.

Agradeço ao Thom Oostendorp pela disponibilização de duas das ferramentas mais importantes utilizadas neste trabalho, as ferramentas *Dipoli* e *QTriplot*.

Agradeço ao Serviço de Neurofisiologia do Instituto Português de Oncologia Francisco Gentil pela facultação dos dados utilizados neste trabalho.

Nada disto era possível sem o amor e carinho das pessoas que se encontram perto de mim. Por isso um agradecimento muito especial aos meus pais por todo o apoio que me têm dado ao longo destes anos, eles têm sido incansáveis e devo-lhes muito.

A todos um grande Obrigado.

Análise Diferenciada das Componentes Independentes de Potenciais Evocados P3a e P3b

Resumo

Este trabalho realizado no âmbito do Mestrado em Imagiologia Médica contribuiu para a continuação do estudo e análise de processos cognitivos utilizando a Análise em Componentes Independentes (ACI) e contribuiu também para o desenvolvimento de uma ferramenta informática onde se encontram implementadas estas técnicas de ACI com a finalidade de facilitar futuros estudos na área da separação "cega" de sinais.

Os dados utilizados neste estudo são respeitantes a registos de potenciais evocados cognitivos $P300$, os quais são respostas cerebrais relacionadas com a atenção. Estes potenciais foram medidos em indivíduos normais e em doentes com epilepsia temporal complexa. O estudo efectuado incidiu no potencial $N100$, também presente nos registos, e nos potencias que constituem o $P300$: o $P3a$ e o $P3b$.

A ACI é um método estatístico implementado computacionalmente que tem como objectivo separar as componentes subjacentes a um conjunto de medidas ou sinais (misturas). Neste trabalho, esta técnica foi aplicada a registos de $P300$, limitando o processamento à procura de apenas algumas componentes. Para a escolha do número de componentes foi utilizado um método chamado Bayesian Information Criterion (BIC).

As componentes calculadas foram, posteriormente, caracterizadas através da aplicação de técnicas de localização dipolar utilizando modelos de dipolo único e de múltiplos dipolos para aproximar as fontes neuronais, e utilizando os modelos esférico e realista para aproximar o meio (cabeça).

Palavras Chave

Análise em Componentes Independentes, Electroencefalografia, Potenciais Evocados, P300, P3a, P3b, BIC, IcaML, Jade, FastIca, pyIca.

Differentiated analysis of the Independent Components of Evoked Potentials P3a and P3b

Abstract

This work, carried through in the scope of the Masters in Medical Imaging contributed for the continuation of the study and analysis of cognitive processes using the Independent Component Analysis (ICA) and contributed also for the development of a computational toolbox implementing the studied ICA techniques, with the purpose of facilitate futures studies in the blind separation of signals area.

The data used in this study is concerned to records of cognitive evoked potentials $P300$, which are related to attention. These potentials were recorded from normal subjects and from patients with temporal complex epilepsy. This study was focused towards the $N100$ potential, also present in the records, and towards the two main components that constitute the $P300$: the $P3a$ and the $P3b$.

The ICA is a statistical and computational method which the main objective is to disclose hidden components that originate sets of measures or signals (mixtures). In this work, this technique was applied to the $P300$ records, finding a limited number of components. For the choice of the number of components to be processed, the Bayesian Information Criterion (BIC) method was used.

The behavior of each component was evaluated through the application of dipole localization techniques using unique and multiple dipole models to approximate the neuronal sources and using spherical and realistic models to shape the environment (head).

Keywords

Independent Component Analysis, Electroencephalography, Evoked Potentials, P300, P3a, P3b, BIC, IcaML, Jade, FastIca, pyIca.

Índice

Página

Lista de Figuras

Página

Lista de Tabelas

Página

Capítulo 1: Introdução

Este trabalho, realizado no âmbito do Mestrado de Imagiologia Médica, teve como finalidade continuar os estudos já realizados na área de potenciais evocados, principalmente o *P*300, estudando as suas principais componentes: o *P*3*a* e o *P*3*b*. Os *P*300 são potenciais evocados cognitivos que permitem estudar os processos cerebrais associados à atenção. Este potencial é positivo e surge com uma latência de cerca de 300ms após a ocorrência de um determinado estímulo. As técnicas utilizadas na análise destes sinais foram as de análise em componentes independentes (ACI) e de localização de fontes eléctricas no cérebro. A ACI tem como objectivo separar o sinal nas diversas componentes que o formam. Nos algoritmos de ACI, o número máximo de componentes nas quais o sinal em estudo é dividido é igual ao número de sensores que foram utilizados na obtenção dos sinais. Quanto ao processo de localização das fontes a partir dos sinais recolhidos tem, neste âmbito, o objectivo de caracterizar a(s) fonte(s) neuronais responsáveis pela actividade eléctrica medida.

Os objectivos deste trabalho são os seguintes:

1. Verificar a existência de problemas enerentes à aplicação da ACI como por exemplo a sub-divisão das componentes em estudo (*overlearning*), neste caso, no estudo dos potenciais *P*3*a* e *P*3*b*.

2. Aplicar diferentes algoritmos de ACI a estes sinais limitando o processamento a apenas algumas componentes, na tentativa de diminuir a sub-divisão das componentes referentes ao potencial evocado *P*300, facilitar a procura destas componentes e remover aquelas que dizem respeito ao ruído ou a artefactos.

3. Classificar as componentes obtidas através de localização dipolar. Esta técnica é habitualmente utilizada para localizar fontes de actividade eléctrica cerebral consideradas pontuais. Neste caso, e uma vez que há evidências no sentido de os geradores subjacentes ao *P*300 envolverem vários circuitos neuronais não necessariamente circunscritos a pequenas regiões, esta técnica será aplicada apenas com o objectivo de parametrizar as componentes.

4. Verificar se existe alguma perda de informação útil com a redução de componentes na ACI, no estudo dos potenciais *P*3*a*, *P*3*b* e *N*100.

5. Verificar se a redução das componentes na aplicação da ACI traz algum benefício ao nível da identificação do tipo de epilepsia, comparando os resultados obtidos com os que foram obtidos num trabalho anterior utilizando os mesmos dados.

6. Construir uma ferramenta informática que implemente as técnicas estudadas com a finalidade de facilitar futuros estudos de separação "cega" de sinais (BSS), para qualquer tipo de sinais (EEG, som, etc). Esta ferramenta deverá ser desenvolvida numa linguagem de fácil utilização e conter rotinas de utilização de acesso livre.

No decorrer deste trabalho foram utilizados dados de Electroencefalografia (EEG) de potenciais evocados reais anteriormente recolhidos.

A metodologia seguida no desenvolvimento deste trabalho é patente na organização do relatório. Após esta introdução, no segundo capítulo são descritos dois tipos de registos usando eléctrodos colocados sobre o escalpe: a electroencefalografia espontânea e o registo de potenciais evocados, sendo este último o que mais interessa. Neste capítulo também são apresentadas algumas noções de fisiologia e de anatomia do sistema nervoso humano. No terceiro capítulo são apresentados os vários tipos de geometrias utilizadas para modelar a cabeça e as características das fontes neuronais utilizadas na localização de geradores neuronais. No quarto capítulo é introduzido o formalismo da análise em componentes independentes e discute-se o desempenho de vários algoritmos que implementam esta técnica. No quinto capítulo são enunciados e comparados os métodos estudados para a estimação do número de fontes presentes num conjunto de misturas. No sexto capítulo são apresentados os resultados obtidos através da aplicação das técnicas enunciadas no capítulo quatro a registos efectuados em indivíduos normais e em doentes. O sétimo capítulo é dedicado à biblioteca pyIca desenvolvida para Python. No último capítulo descrevem-se conclusões sobre os estudos efectuados, tecem-se alguns comentários finais e são feitas propostas de trabalhos futuros.

Em anexo, inclui-se informação relevante para a utilização deste trabalho em futuros projectos. Nos anexos A, B e C são explicados com maior pormenor os três algoritmos de ACI estudados: o FastIca, o Infomax (IcaML) e o Jade. No anexo D é descrito o método BIC (*Bayesian Information Criterion*). No anexo E é apresentada a definição de cumulante de uma distribuição. Nos anexos F e G é apresentada a documentação relativa às bibliotecas icapp e pyIca, respectivamente. Finalmente nos anexos H, I e J são colocadas as listagens do código desenvolvido especificamente para este trabalho.

Capítulo 2: Electroencefalografia e Potenciais Evocados

2.1 Introdução

Neste capítulo é feita uma introdução a dois tipos de registos usando eléctrodos colocados sobre o escalpe para medir a actividade eléctrica do cérebro: a electroencefalografia espontânea e o registo de potenciais evocados. Antes de passar à explicação destas técnicas, introduzem-se algumas noções fisiológicas e anatómicas do sistema nervoso humano, para melhor se entender o seu funcionamento.

2.2 Elementos de Neurofisiologia

O sistema nervoso central é constituído por dois tipos diferentes de células: os neurónios e as células gliais, cada um dos quais associado a diferentes funções, como se poderá consultar em diversos livros de texto [53][48].

2.2.1 Células do Sistema Nervoso

Os neurónios são a unidade básica do sistema nervoso. O funcionamento dos neurónios passa pela geração de sinais eléctricos que são transferidos de uma parte da célula para outra parte da célula, e pela libertação de neurotransmissores que permitem a transmissão de informação para outras células. O neurónio é visto como um integrador devido ao facto da sua saída reflectir o balanço das entradas que lhe são aplicadas pelas outras células a que está ligado. Os neurónios aparecem no sistema nervoso sob diversas formas e tamanhos. Apesar da existência desta diversidade, os neurónios são, de forma resumida, constituídos por quatro partes: o corpo celular, as dendrites, o axónio e os terminais do axónio. A estrutura de um neurónio típico pode ser vista na figura 2.1.

As dendrites e o corpo celular são as partes do neurónio que recebem e processam (integram) a informação recebida de outros neurónios, informação essa que poderá ser no sentido da excitação ou no sentido da inibição, ou seja, no sentido de gerar sinais e transmiti-los para outros neurónios ou no sentido de evitar a sua criação. O axónio tem como função transportar a informação processada para outros

neurónios, sendo alguns deles cobertos por secções de mielina separadas por nódulos chamados de nódulos de Ranvier. A mielina aumenta a velocidade de condução dos sinais eléctricos ao longo do axónio conservando a sua energia [48].

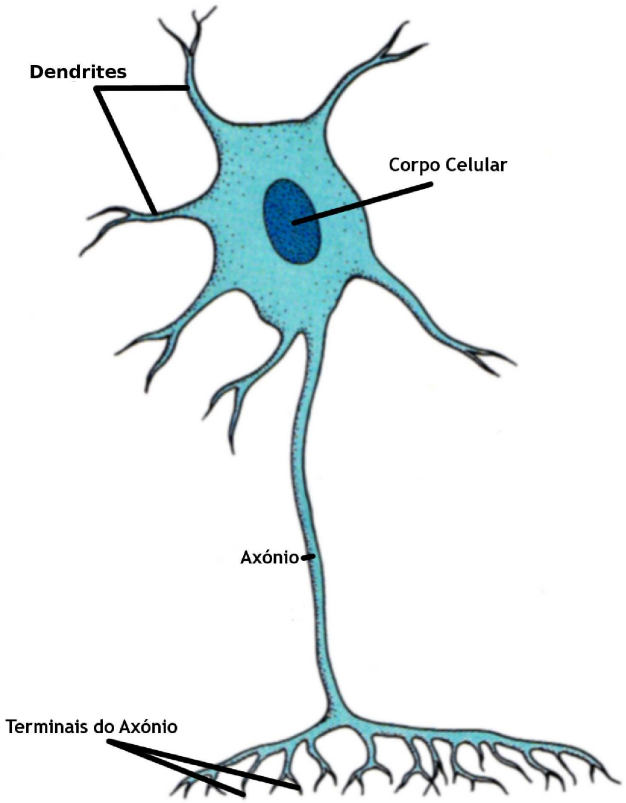


Figura 2.1: Estrutura dos neurónios (adaptado de [53]).

Do ponto de vista da propagação do sinal, os neurónios são classificados de três formas [53]:

1. Neurónios aferentes que transmitem informação dos receptores para o sistema nervoso central.
2. Neurónios eferentes que transmitem informação do sistema nervoso central para células efectoras (células musculares ou outros neurónios).
3. Os inter-neurónios que se encontram inteiramente no sistema nervoso central e formam circuitos entre neurónios ou ligam neurónios aferentes aos neurónios eferentes ou vice-versa.

Embora os neurónios sejam a unidade base do sistema nervoso, estes representam apenas 10% da totalidade de células nele presentes [53]. As restantes células são chamadas de células gliais e têm como principais funções regular a composição do fluído extracelular, formar a mielina, servir de guia para o desenvolvimento dos neurónios e fornecer funções de imunidade.

2.2.2 Potencial de Repouso

Todas as células em estado de repouso têm uma diferença de potencial transmembranar, existindo um potencial negativo no interior da célula em relação ao seu exterior. Este potencial é chamado de potencial de repouso de membrana e tem um valor típico de -70 mV [53].

O potencial de repouso, uma vez que é gerado pela difusão de determinados iões através da membrana, é particularmente influenciado pela permeabilidade da membrana a esses iões. O sentido da difusão é determinado pelas diferenças de concentração de alguns iões específicos nos meios intra e extracelulares

e pelo sinal da diferença de potencial estabelecida. Tipicamente existe um excesso de iões de sódio Na^+ no meio extracelular e um excesso de iões de potássio K^+ no meio intracelular.

No estado de repouso, os canais abertos através da membrana são predominantemente permeáveis ao ião K^+ tornando assim a membrana celular mais permeável ao K^+ do que ao Na^+, e, de acordo com o gradiente de concentração, os iões K^+ vão fluir do meio intracelular para o meio extracelular tornando o interior da célula negativo em relação ao seu exterior. O potencial evolui, negativamente, em direcção ao potencial de equilíbrio do K^+ que é de -90 mV [53]. No entanto, com a presença de canais de Na^+ abertos na membrana celular, alguns iões de Na^+ vão fluir para o meio intracelular, impedindo assim que o potencial de equilíbrio de K^+ seja atingido, observando-se, então, uma ligeira diminuição no valor absoluto da diferença de potencial (ver figura 2.2).

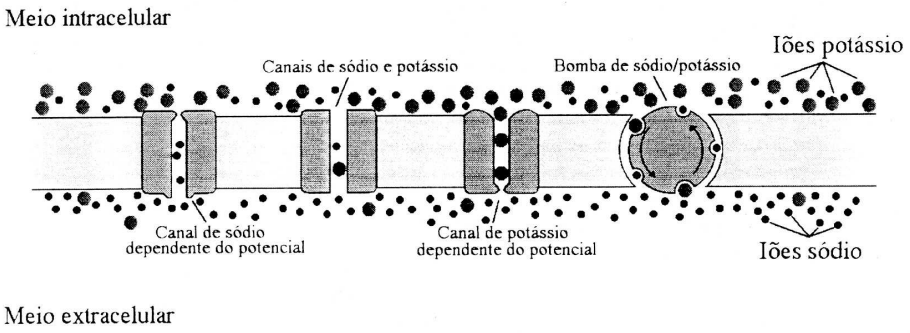


Figura 2.2: Transferências de iões através da membrana neuronal (adaptado de [55] citado em [50]).

A contínua entrada de iões Na^+ para o interior da célula e a contínua saída de iões K^+ para o exterior da célula iriam alterar as concentrações intra e extracelulares levando o potencial da célula a aumentar. Esta situação não chega a acontecer (no estado repouso) devido à existência de mecanismos de transporte activo na membrana. Esses mecanismos são bombas de sódio/potássio que deslocam três iões Na^+ para fora da célula por cada dois iões K^+ deslocados para dentro da célula. Este transporte activo permite manter a concentração de iões Na^+ baixa e a concentração de iões K^+ alta no meio intracelular, consumindo, portanto, energia. Desta forma, o potencial de repouso é mantido a cerca de -70 mV [53].

2.2.3 Potencial de Acção

O potencial de acção é uma rápida mudança no potencial da membrana celular durante o qual a célula rapidamente despolariza e repolariza. O potencial de acção ocorre devido à existência de canais Na^+ e K^+ na membrana que são dependentes do potencial da membrana [53].

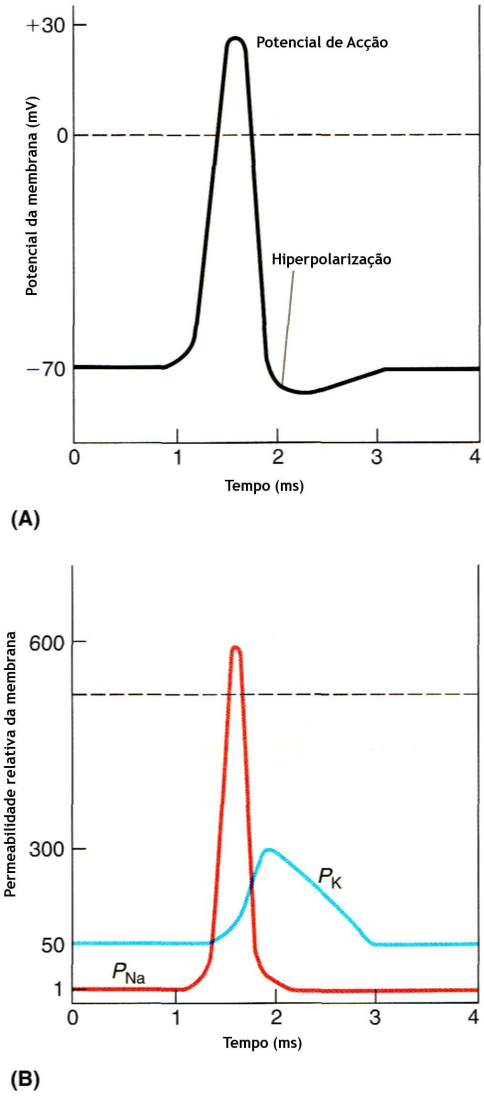


Figura 2.3: Morfologia do sinal do potencial de acção relacionada com a permeabilidade relativa da membrana celular (adaptado de [53]).

Quando a célula, polarizada, sofre uma despolarização (por via de um estímulo) de duas ou três dezenas de mV, atingindo o patamar de aproximadamente −55 mV, esta despolarização vai ser acentuada com a abertura de mais canais de sódio que permitem a entrada destes iões de sódio para dentro da célula ([32] citado em [50]). Na realidade, são abertos mais canais de sódio e potássio mas como os canais de sódio abrem mais rapidamente que os canais de potássio, esta situação provoca uma realimentação positiva, aumentando a concentração dos iões de sódio dentro da célula e elevando o potencial na direcção do potencial do equilíbrio do sódio que é de +60 mV [53]. O potencial de equilíbrio do sódio não chega a ser atingido devido à abertura dos canais de potássio, que permitem o fluxo destes iões para fora da célula o qual, se opõe, à subida do sinal. A abertura de mais canais de potássio conjuntamente com a circunstância de os canais de sódio fecharem rapidamente, são responsáveis pela repolarização da membrana. Como existe um maior número de canais potássio, e estes fecham lentamente, vai ocorrer uma hiperpolarização que terminará quando estes canais adicionais fecharem. À medida que as diferentes partes da membrana celular vão sendo despolarizadas, o potencial de acção é transportado através da célula propagando-se como uma onda.

2.2.4 Sinapses

Os potenciais de acção providenciam transmissão de informação a longa distância no sistema nervoso, passando informação a outros neurónios através das sinapses. A sinapse é uma junção anatomicamente especializada entre dois neurónios onde a actividade eléctrica de um neurónio (pré-sináptico) influencia a actividade eléctrica do outro neurónio (pós-sináptico). Quando activo, o potencial de membrana do neurónio pós-sináptico pode-se aproximar do limiar para a geração de um potencial de acção através de uma sinapse excitatória, ou afastá-lo do limiar através de uma sinapse inibitória [53]. A sinapse excitatória vai despolarizar a célula pós-sináptica e a sinapse inibitória vai hiperpolarizar a célula pós-sináptica.

A transmissão de informação poderá ser convergente no caso de várias células pré-sinápticas afectarem uma célula pós-sináptica e poderá ser divergente no caso de uma célula pré-sináptica influenciar várias células pós-sinápticas [53].

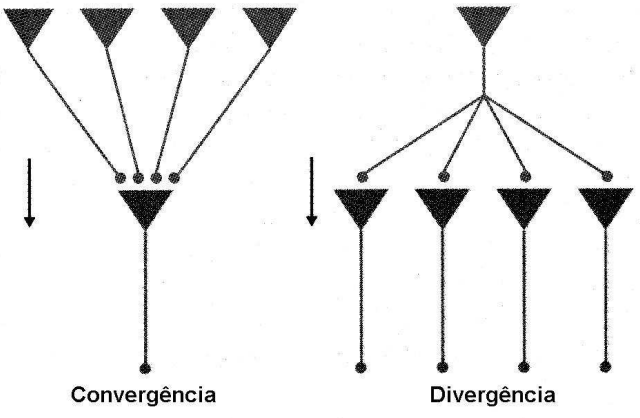


Figura 2.4: Transmissão de informação convergente e divergente (adaptado de [53]).

A geração do potencial de acção pela célula pós-sináptica vai depender do número de sinapses activas e do número de sinapses inibitórias ou excitatórias. Existem dois tipos de sinapses, as eléctricas e as químicas. Nas primeiras, o potencial de acção da célula pré-sináptica é passado directamente para a célula pós-sináptica através das ligações físicas entre as células. O potencial vai provocar a despolarização da célula pós-sináptica até ao limiar e assim gerar um potencial de acção na célula pós-sináptica. As sinapses eléctricas são muito localizadas e raras nos mamíferos [53].

Nas sinapses químicas, a transmissão de informação é feita através da libertação de neurotransmissores pela célula pré-sináptica. Quando os sinais atingem o terminal do axónio, os neurotransmissores são libertados para o espaço entre as células pré e pós-sináptica chamada de fenda sináptica. Na célula pós-sináptica encontram-se receptores sensíveis aos neurotransmissores químicos. Os receptores, ao

detectarem os neurotransmissores vão induzir fluxos iónicos que alteram a polarização da membrana da célula pós-sináptica. A alteração de potencial provocada pela sinapse na célula pós-sináptica é chamada de potencial pós-sináptico [53].

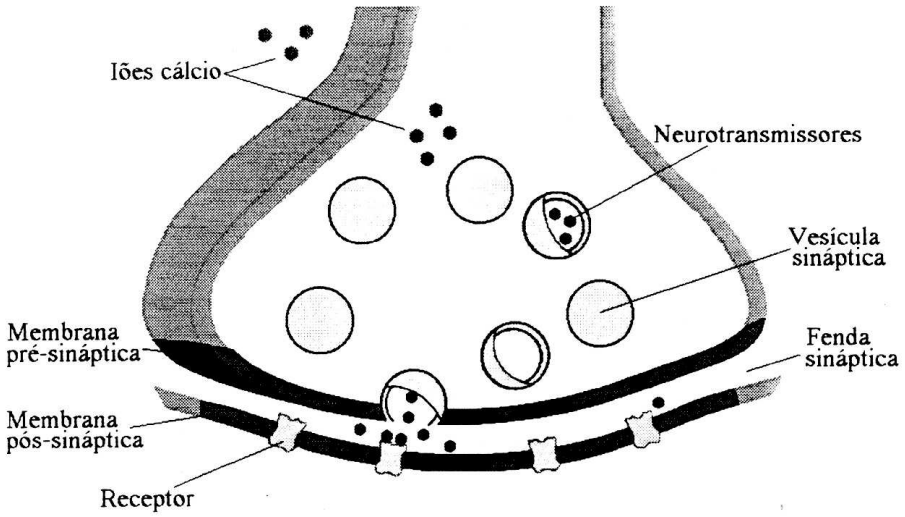


Figura 2.5: Esquema de uma sinapse química. O potencial de acção ao chegar ao terminal do axónio da célula pré-sináptica vai desencadear a entrada de iões Ca^{++} que por sua vez vão libertar os neurotransmissores que se encontram dentro das vesículas sinápticas (adaptado de [30] citado em [50]).

O potencial de acção, funcionando como uma onda, propaga-se pelo axónio e atinge o terminal do axónio. Cada potencial de acção que atinge o terminal do axónio da célula pré-sináptica pode provocar um potencial pós-sináptico de aproximadamente 1mV ([13] citado em [50]). Como a despolarização necessária para atingir o patamar de criação de um novo potencial de acção na célula pós-sináptica é de algumas dezenas de mV, é necessário a soma de vários potenciais pós-sinápticos excitatórios para provocar uma despolarização suficiente para o desencadear de um novo potencial de acção na célula pós-sináptica ([45] citado em [50]).

Cada neurónio estabelece centenas de sinapses em que a informação, excitatória ou inibitória, é recebida e é integrada. O comportamento do neurónio resultará dessa integração. Essa integração, que é espacial e temporal, é responsável pela duração dos potenciais pós-sinápticos. A duração dos potenciais pós-sinápticos pode ser na ordem do segundo, até vários minutos ([45] citado em [50]).

2.3 Anatomia do Sistema Nervoso

O sistema nervoso é dividido em duas partes: o sistema nervoso central (SNC) que é constituído pelo encéfalo e pela espinal medula, e pelo sistema nervoso periférico (SNP).

2.3.1 Sistema Nervoso Central

2.3.1.1 Medula Espinal

A medula espinal passa pela coluna vertebral e é composta por substância cinzenta no centro que é rodeada por substância branca. A substância cinzenta é constituída pelos corpos celulares dos neurónios e a substância branca é constituída pelas fibras nervosas (axónios e dentrites). A cor branca deve-se à camada de mielina que reveste as fibras [53].

As fibras nervosas percorrem longitudinalmente a medula espinal, algumas de forma descendente para levar a informação do encéfalo para o sistema nervoso periférico, e outras de forma ascendente para levar a informação até ao encéfalo [53].

2.3.1.2 Encéfalo

O encéfalo está protegido pela caixa craniana, por membranas finas chamadas meninges e pelo líquido céfalo-raquidiano. As meninges são a dura-máter (junta ao osso), a pia-máter (junto ao tecido nervoso) e a aracnóideia (no meio das outras duas). No espaço existente entre a aracnóideia e a pia-máter encontra-se o líquido céfalo-raquidiano [53] (ver figura 2.6).

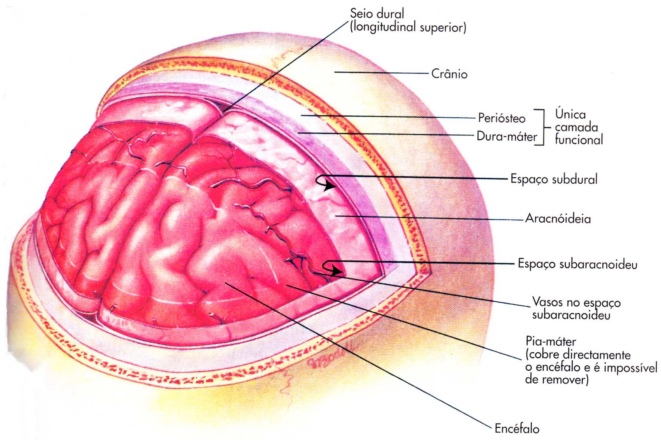


Figura 2.6: Meninges protectoras do encéfalo [48].

As meninges, além de protegerem o encéfalo e a medula espinal, fazem circular e absorvem o líquido céfalo-raquidiano. Este liquido é produzido e depositado no sistema ventricular (ventrículos interligados) [48]

O encéfalo é composto por cinco sub-divisões: cérebro, tronco cerebral, sistema límbico, diencéfalo e cerebelo (ver figura 2.7) [53]:

- O cerebelo desempenha um papel importante na manutenção do equilíbrio e na coordenação da actividade motora.
- O diencéfalo é constituído pelo tálamo e pelo hipotálamo. O tálamo é a zona onde chegam a maior parte das fibras nervosas sensitivas e é aqui as informações sensoriais são retransmitidas para as respectivas áreas do córtex cerebral. O hipotálamo desempenha um papel na regulação da temperatura do corpo, da fome, da sede, do sistema reprodutor e da circulação sanguínea. O hipotálamo tem também como função a geração e regulação do ritmo cardíaco.
- O sistema límbico é responsável pela geração de emoções e comportamento emocional. Também desempenha um papel importante na aprendizagem.
- O tronco cerebral é constituído pelo mesencéfalo, protuberância anular e pelo bolbo raquidiano. O mesencéfalo é uma zona com funções no processamento de informação visual e auditiva. O bolbo raquidiano é o ponto de passagem dos nervos que ligam a medula ao cérebro. Contém grupos de neurónios especializados na regulação de funções fisiológicas vitais tais como o ritmo cardíaco, a respiração e a pressão arterial. Esta região também influencia o sono e a tosse.

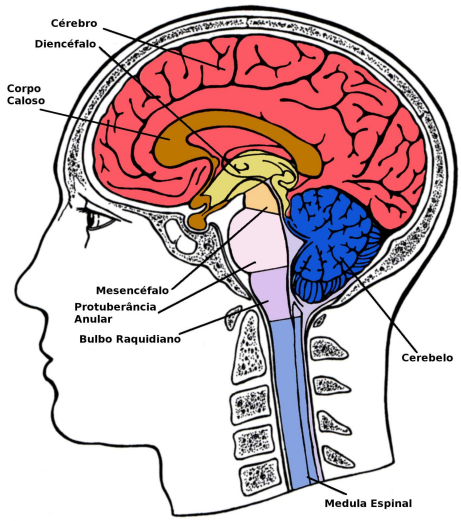


Figura 2.7: Alguma estruturas anatómicas do encéfalo do encéfalo (adaptado de [53]).

- O cérebro é dividido nos hemisférios esquerdo e direito, que estão unidos pelo corpo caloso. O corpo caloso é pois um agrupamento de fibras nervosas que estabelecem a ligação entre os dois hemisférios [53]. Existem em cada hemisfério quatro lobos (ver figura 2.8(a)) [48]:

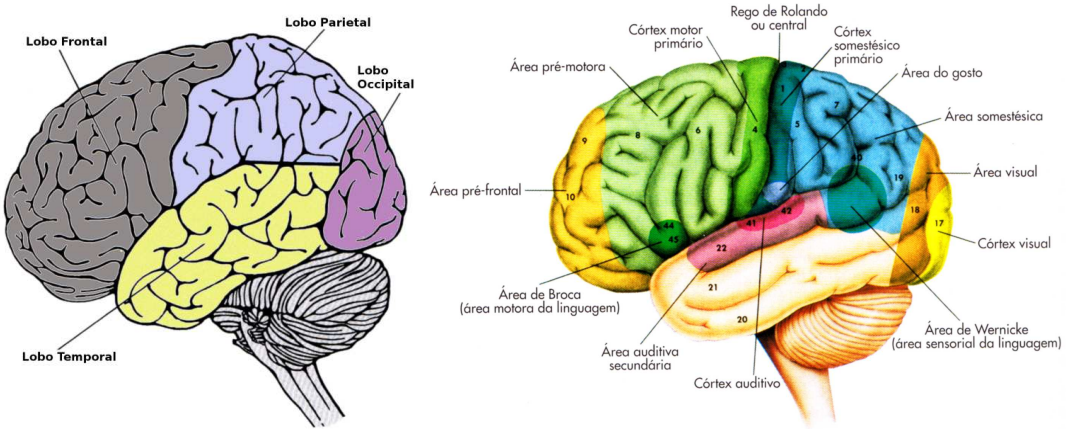
  Lobo frontal relacionado com o olfacto, motivação, humor, agressão e controle da mobilidade voluntária.

  Lobo parietal contém as principais áreas sensitivas que recebem a maioria dos estímulos sensoriais, paladar e equilíbrio.

  Lobo occipital contém os centros visuais.

  Lobo temporal este lobo recebe os estímulos olfactivos e auditivos. Ele está também implicado no pensamento abstracto, capacidade de julgamento e memória.

  Cada hemisfério é constituído por uma camada de substância branca e outra de substância cinzenta chamada córtex. A figura 2.8(b) mostra uma vista do córtex cerebral esquerdo, onde estão apresentadas algumas áreas funcionais. As vias sensoriais projectam-se para as áreas sensitivas primárias no córtex cerebral (córtex primário) enquanto que as áreas secundárias (associativas), que estão adjacentes às áreas primárias, interpretam os estímulos das áreas primárias (ver figura 2.8(b)) [48].



(a) Lobos cerebrais. (b) Áreas funcionais do córtex.

Figura 2.8: a) Lobos Cerebrais (adaptado de [53]). b) Áreas funcionais do córtex cerebral esquerdo [48].

2.3.2 Sistema Nervoso Periférico

O SNP é constituído por um conjunto de nervos (axónios) que se estendem a partir do sistema nervoso central (principalmente a partir da espinal medula). O conjunto compreende 43 pares de nervos: 12

pares de nervos cranianos e 31 pares de nervos espinais. A maioria destes nervos contém axónios de neurónios aferentes e eferentes. Desta forma, o SNP é constituído por duas divisões: a divisão aferente e a divisão eferente. Por sua vez, a divisão eferente é subdividida nas partes somáticas e autónomas [53]. Enquanto que o sistema nervoso somático enerva os músculos esqueléticos, o sistema nervoso autónomo é o sistema nervoso automático. Este último actua independente da nossa vontade, enerva os órgãos, músculos cardíacos, glândulas e trato gastrointestinal, auxilia o controle da pressão arterial, secreção gastrointestinal, temperatura corporal e em muitas outras funções [53].

2.4 Medição da Actividade Eléctrica Cerebral

Foram referidos dois potenciais que são susceptíveis de serem medidos no escalpe: o potencial de acção e o potencial pós-sináptico [37] citado em [50]. Devido à curta duração temporal e pouca extensão no espaço, os potenciais de acção pouco contribuem para o registo da actividade eléctrica medida a nível do escalpe. Assim, assume-se que apenas os potenciais pós-sinápticos contribuem para os sinais medidos através da electroencefalografia. Os potenciais pós-sinápticos derivam da soma temporal e espacial da actividade de todas as sinapses que chegam a um conjunto de neurónios envolvidos na mesma função, ou seja, ocupam mais espaço e prolongam-se por mais tempo e devido a estes factores estes potenciais são mais facilmente mensuráveis em locais afastados da zona onde são formados. A actividade eléctrica associada às células não é suficiente para permitir a medição dos sinais ao nível do escalpe. O perfil da disposição dos neurónios também é importante. As células piramidais que se encontram em algumas camadas corticais seguem uma geometria chamada paliçada. Esta geometria implica que os neurónios se apresentem paralelos uns aos outros e perpendiculares à superfície do córtex (ver figura 2.9). Se existir uma actividade síncrona nestes neurónios, os efeitos das correntes geradas pelos potenciais pós-sinápticos somam-se, facilitando, desta forma, o seu registo através da electroencefalografia [50].

2.5 Técnicas de Registo da Actividade Eléctrica Cerebral

Existem dois tipos de registos usando eléctrodos colocados sobre o escalpe para medir a actividade eléctrica do cérebro: a electroencefalografia espontânea, que regista a actividade espontânea do cérebro e o registo de potenciais evocados. Em seguida é feita uma breve introdução sobre as técnicas para a obtenção destes registos.

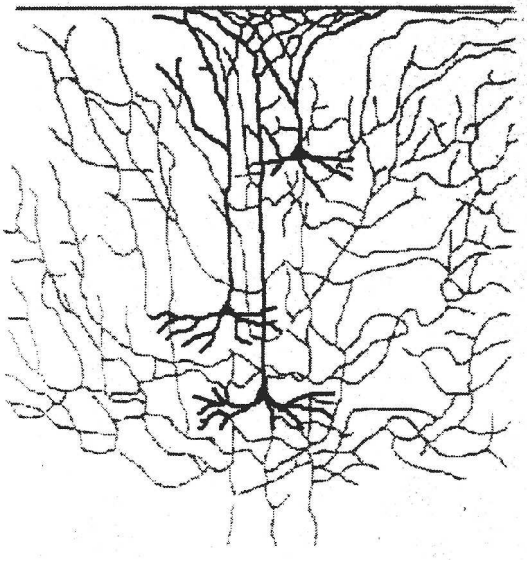


Figura 2.9: Neurónios piramidais corticais ilustrativos da geometria em paliçada [42].

2.5.1 Eléctrodos

Os sinais eléctricos são recolhidos através do uso de eléctrodos colocados sobre o escalpe. Os eléctrodos são feitos de materiais bons condutores (metais) para permitir a medida dos sinais. Existem diversas formas de eléctrodos. A forma dos eléctrodos vai depender do local e do modo como serão utilizados . Os eléctrodos mais utilizados têm forma circular com ou sem orifício [46].

É necessário assegurar o bom contacto entre o escalpe e os eléctrodos, para isso utiliza-se um gel electrolítico que também ajuda a baixar a impedância. A impedância de todos os eléctrodos deverá ser a mesma e inferior a 5000Ω, sendo valores normais impedâncias abaixo dos 3000Ω ou até mesmo 1000Ω [46]. A impedância dos eléctrodos deverá também ser adequada à impedância de entrada dos amplificadores do electroencefalógrafo. Esta condição é necessária para que não haja perdas no sinal ao entrar nos circuitos amplificadores [46].

A colocação dos eléctrodos segue um padrão estabelecido internacionalmente que é conhecido como Sistema Internacional 10-20 [46] e é apresentado na figura 2.10.

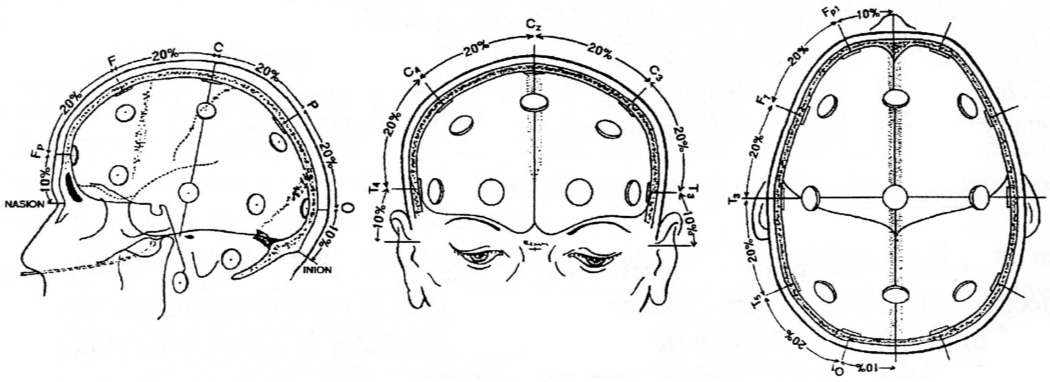


Figura 2.10: Sistema internacional 10-20 (adaptado de [27] citado em [50]).

Os eléctrodos são identificados através de letras que indicam a área do cérebro que cobrem. As letras são Fp, F, C, P, T, O e A que denotam as áreas Frontopolar, Frontal, Central, Parietal, Temporal, Occipital e Auricular, respectivamente. Os eléctrodos são também identificados por dígitos que têm como função numerar e distinguir os eléctrodos da mesma região cortical. Estes designam também o lado do cérebro em que vão recolher dados, sendo os números pares referentes ao hemisfério direito e os números ímpares referentes ao hemisfério esquerdo. O z refere-se à zona central [46].

Os números 10-20 referem-se às percentagens de separação entre cada local de colocação de eléctrodos em relação aos pontos de referência básicos. Estes últimos correspondem ao nasion que é localizado por baixo da testa imediatamente por cima do nariz e o inion que é uma protuberância óssea localizada a meio da nuca e os pontos pré-auriculares que estão localizados em frente do canal auditivo.

2.5.2 Montagens

Na montagem dos eléctrodos, o aspecto mais importante a ter em conta é a escolha do tipo de referência para fazer o registo das diferenças de potencial. As montagens geralmente utilizadas em EEG dividem-se em duas categorias [46]:

Monopolar - Nesta montagem o potencial eléctrico recebido por cada um dos eléctrodos é comparado com o potencial registado por um ou mais eléctrodos de referência. A principal vantagem desta montagem é a fácil comparação de registos entre diferentes pontos e tem como desvantagem o facto se alguma actividade parasita afectar o eléctrodo de referência (uma vez que nenhum eléctrodo é suficientemente isento de actividade eléctrica), esta vai repercutir-se nos registos de todos os outros eléctrodos. Outra referência utilizada nas montagens monopolares é a média dos valores medidos nos eléctrodos. Neste caso o potencial eléctrico recebido por cada um dos eléctrodos é comparado com uma referência que é calculada a partir da média dos potenciais

medidos em todos os eléctrodos. As várias contribuições tendem a anular-se originando um ponto de referência estável. Esta montagem resolve o problema de repercussão de artefactos, mas atenua os picos de actividade localizados em regiões restritas.

Bipolar - Nesta montagem as diferenças de potencial são calculadas entre dois eléctrodos consecutivos. Esta montagem tem como vantagem resolver o problema de repercussão de artefactos da montagem monopolar e realçar regiões com gradientes de potenciais elevados. Apresenta como desvantagem a dificuldade da interpretação imediata dos resultados.

Com o aparecimento dos sistemas de EEG digitais (que serão discutidos na secção seguinte), o operador passou a dispor dos dados em formato digital. Este facto tornou possível a fácil alteração da montagem e pesquisar a existência de alguma informação que possa ter-se tornado visível após essa alteração sem ter que repetir o exame [46]. Por exemplo, se um registo for obtido utilizando uma montagem monopolar utilizando um eléctrodo de referência, é depois possível visualizar o registo em montagem monopolar utilizando como referência a média dos eléctrodos, ou, em montagens bipolares, bastando para isso calcular a diferença entre os eléctrodos.

Na interpretação do EEG e no estudo de potenciais evocados, é útil tentar reduzir ou eliminar a actividade eléctrica de um ou dois pontos para que a descarga eléctrica observada reflicta melhor a actividade de um ponto em particular [46]. Com o objectivo de melhorar a localização da actividade focal no escalpe, Hjorth introduziu um método que passa por sujeitar os dados registados à montagem laplaciana [42][46]. Esta montagem utiliza o operador matemático laplaciano com o objectivo de realçar e isolar a actividade que é única a um determinado eléctrodo. Este método aumenta a possibilidade de se fazer várias interpretações e visualizar a actividade isolada de vários pontos com apenas uma montagem.

Quanto ao número de eléctrodos, na localização de fontes, quanto maior for o número de eléctrodos, maior será a precisão na localização das fontes [44][50]. O maior número de eléctrodos tem também a vantagem de fornecer posições extra que providenciam um degrau extra de redundância que pode ajudar na decisão se uma determinada alteração de um potencial evocado é "real", ou trata-se de ruído ou de algum artefacto [33].

2.5.3 Sistemas Digitais de EEG

Os primeiros sistemas de registo eram sistemas analógicos e permitiam apenas o armazenamento dos dados (sinais) em papel. Isto limitava o número de eléctrodos a serem registados em simultâneo, dificultava a mudança de montagem dos eléctrodos e não permitia um estudo mais aprofundado como por exemplo estudos no domínio da frequência. Com o aparecimento do EEG digital, a aquisição e a análise

dos dados recolhidos são feitas de um modo digital. Deste modo, os sinais analógicos recolhidos são amostrados e quantizados. Estes dados podem ser armazenados em suportes digitais, permitindo assim o uso de sistemas informáticos para ajudar no estudo e análise dos sinais obtidos. Os equipamentos de EEG digital começaram a ser equipados com um módulo logístico informático completo que permite fazer a análise dos sinais recolhidos. Este módulo permite a aplicação de técnicas de cálculo de médias, de filtros digitais, da Transformada de Fourier para o estudo das frequências, de métodos estatísticos etc., tudo isto através do *software* instalado [24].

Com a digitalização da informação, surgiu uma nova técnica: a topografia cerebral (ver figura 2.11). Através de um programa instalado no módulo informático do aparelho de EEG, consegue-se, nesta técnica, visualizar, através de mapas coloridos ou escala de cinzentos a actividade eléctrica no escalpe [24]. Esta imagem é obtida através da codificação da amplitude eléctrica registada em cada eléctrodo na forma de gradientes de cor. Os pontos de imagem que ficam entre as posições dos eléctrodos são calculados através de técnicas de interpolação (como por exemplo a interpolação *Spline* multi-dimensional [19]), e usam como referência os valores dos eléctrodos vizinhos.

A representação topográfica da actividade eléctrica permite uma melhor localização de alterações na amplitude como também ajuda na percepção de relações espaciais entre diferentes registos. A partir de imagens topográficas cerebrais obtidas ao longo do tempo, é possível produzir animações que permitem fazer o estudo dinâmico da função cerebral [24], explorando a grande resolução temporal característica da técnica da EEG [50].

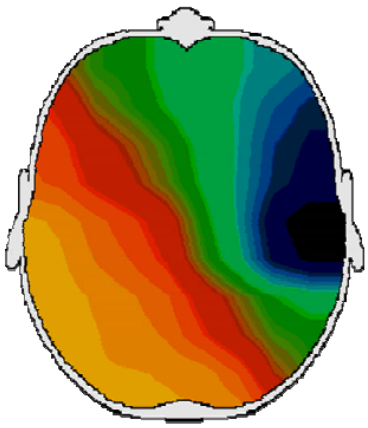


Figura 2.11: Mapa topográfico cerebral [50].

2.6 Potenciais Evocados

2.6.1 Definição

Os potenciais evocados (PE), também conhecidos por respostas evocadas, são potenciais eléctricos gerados pela activação de populações neuronais desencadeada por estímulos específicos provocados no indivíduo em estudo, e que se sobrepõem à actividade espontânea registada [51]. Esses potenciais são característicos dos estímulos aplicados e podem ser a resposta a um ou mais estímulos. Os estímulos podem ser visuais, auditivos ou sensoriais.

A medição de potenciais evocados tem-se revelado uma importante ferramenta ao nível do diagnóstico de distúrbios neurológicos. Diversos estudos têm feito comparações entre componentes de registos de PE pertencentes a indivíduos normais e pertencentes a indivíduos doentes. Estes estudos comprovaram desvios entre os dois grupos ao nível das latências e das amplitudes das componentes comparadas [51]. São estas diferenças que permitem o diagnóstico de determinadas patologias.

2.6.2 Características

Os PE são sinais de menor energia (amplitude) que a observada num EEG normal (actividade espontânea). Os PE apresentam uma amplitude mínima de 0.5μV enquanto que a actividade espontânea apresenta tipicamente uma amplitude de 10-30μV [50]. Por este motivo, para que o potencial evocado em estudo se destaque da actividade espontânea, fazem-se diversos registos da actividade eléctrica após a aplicação repetida do mesmo estímulo ou conjunto de estímulos externos (considerando-se desta forma que a actividade espontânea é similar para cada um dos ensaios) [51]. Recorrendo a técnicas de cálculos de médias, atenua-se o ruído de fundo da actividade cerebral espontânea. As componentes (picos) características que conferem a morfologia do potencial evocado em estudo tornam-se mais pronunciadas. Os efeitos do cálculo da média dos sinais podem ser vistos na figura 2.12.

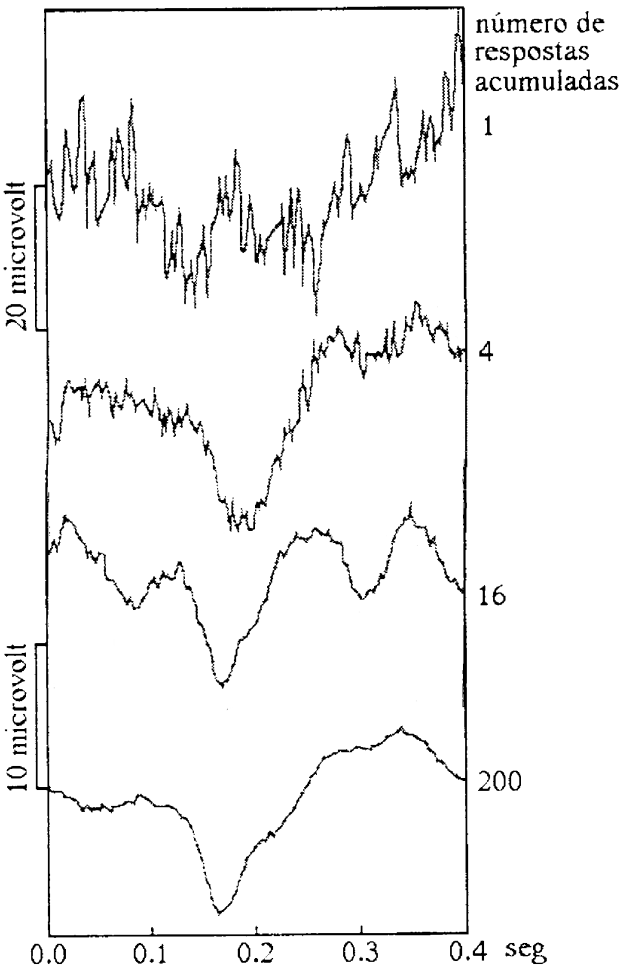


Figura 2.12: Médias de registos de um potencial evocado visual a partir de vários ensaios [50]. Com o aumento do número de registos através de vários ensaios, verifica-se que a média dos registos vai tornando mais clara a presença do potencial.

Os registos de PE são constituídos por picos consecutivos (positivos ou negativos) chamados de componentes que estão relacionadas com as etapas de processamento de informação e são caracterizadas pelas suas latências [33] e pelas suas amplitudes. Entende-se por latência o tempo que separa o estímulo do aparecimento de uma componente. A análise dos potenciais evocados passa pela identificação destas componentes. As componentes são designadas na forma Nx para picos negativos e Px para picos positivos, ambos em relação à referência. O x refere-se à latência do pico em milissegundos. Na figura 2.13 é apresentado um registo com várias componentes.

As componentes (potenciais) podem ser classificadas de curta latência (ou exógenas) se estas ocorrerem entre os 80 e os 200ms, e, de longa latência (ou endógenas) se tiverem tempos de ocorrência superiores aos 200ms [33]. As componentes de curta latência estão essencialmente relacionadas com respostas físicas mais imediatas do organismo e são, por vezes, referidas como componentes "obrigatórias" [33]. As componentes de longa latência envolvem processos mais complexos (processos cognitivos) relacionados com a interpretação dos estímulos [33].

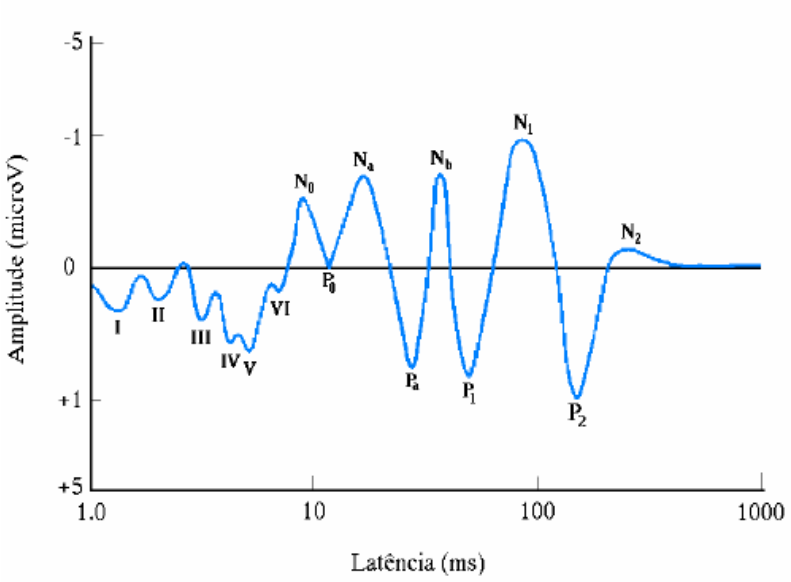


Figura 2.13: Várias componentes de um potencial evocado auditivo (adaptado de [45] citado em [50]).

2.6.2.1 Potenciais Evocados Cognitivos (componente P300)

O *P*300 é um potencial (componente) caracterizado por um pico positivo, que, num indivíduo saudável, ocorre na zona dos 300ms após a estimulação e está relacionado com processos de cognição e atenção [28]. Este potencial evocado é observado quando o paciente discrimina eventos que diferem entre si num dado parâmetro [28]. No caso de potenciais evocados auditivos *P*300, pede-se ao indivíduo que distinga um som infrequente de um outro frequente.

A componente *P*300 é sempre antecedida por outras componentes, o *N*100 e, por vezes, o *P*200. O *N*100 tem máximos negativos nas zonas centro-frontal e surge sempre, mesmo sem colaboração do individuo examinado, estando esta componente relacionada com respostas físicas imediatas [33][47].

A latência e amplitude do *P*300 apresentam-se alteradas em doentes com deterioração mental [28]. O estudo dos potenciais *P*300 pode ser útil no diagnóstico de disfunções mentais como a doença de Alzeimer, ou no estudo de desordens clínicas como o alcoolismo e a esquizofrenia [28].

O *P*300 é composto por duas sub-componentes independentes que se adicionam e que formam o pico *P*300: *P*3*a* e *P*3*b*. Estas componentes apresentam características diferentes que também explicam, de certa forma, a grande latência do *P*300. O *P*3*b* tem normalmente uma distribuição centro-parietal enquanto que o *P*3*a* tem uma distribuição mais frontal [28]. É aceite que o *P*3*b* está mais relacionado com o processamento da informação, com a actualização da memória e do estado de expectativa [28] e que o *P*3*a* corresponde mais a uma resposta orientada, a processos de alerta imediato e é de habituação mais rápida. [28]

O estudo que vai ser feito neste trabalho incidirá principalmente nos potenciais evocados cognitivos $P300$ (e suas componentes $P3a$ e $P3b$) e no $N100$. Este estudo irá ser feito recorrendo à análise em componentes independentes dos registos destes potenciais, numa tentativa de recuperação das fontes que os produziram (separação das componentes).

2.6.3 Epilepsia

A epilepsia é uma doença neurológica associada a descargas sincronizadas anormais de neurónios cerebrais. Estas descargas reflectem-se nos registos eletroencefalográficos através de picos com maiores amplitudes comparativamente aos registos de indivíduos saudáveis [53]. A grande variedade de tipos de epilepsia dificulta a sua classificação. A classificação que é apresentada em seguida foi encontrada em [39]. As epilepsias distinguem-se como focais (circunscritas a determinadas regiões do córtex) e generalizadas (abrangendo todo o córtex).

As crises focais subdividem-se em:

- Crises parciais simples: ocorrem sem a perda de consciência e podem ter características motoras, sintomas somato-sensoriais, autonómicos ou psíquicos.
- Crises parciais complexas: podem ter início focal seguindo-se perda de consciência ou perda de consciência coincidindo com o início da crise.
- Crises parciais que evoluem para generalizadas: podem ter início parcial simples e evoluírem para generalizadas, podem ter início parcial complexo e evoluírem para generalizadas, ou podem ter início parcial simples, evoluírem para parcial complexo e, em seguida, tornarem-se generalizadas.

As crises generalizadas subdividem-se em:

- Crises de ausência.
- Crises clónicas caracterizadas por contracções musculares repetitivas.
- Crises mio-clónicas caracterizadas por contracções musculares simétricas da cara ou dos membros.
- Crises tónicas caracterizadas por rigidez muscular generalizada.
- Crises tónico-clónicas.
- Crises atónicas caracterizadas por perda curta e generalizada da tonicidade muscular.

Como a epilepsia está associada a descargas eléctricas cerebrais anormais, estas podem ser diagnosticadas através da electroencefalografia. São normalmente realizados dois tipos de registos [40]: O registo interictal (realizado entre crises) e o registo ictal (registo da crise). A maioria dos registos realizados são os interictais devido ao facto de que no registo ictal, é necessário induzir uma crise, e, durante essa crise, os registos são normalmente contaminados por artefactos resultantes de movimentos involuntários.

Nos casos de epilepsia focal, a posição dos eléctrodos onde se verifica a actividade anormal pode indicar a localização da região afectada. Esta informação pode ser útil na identificação do tipo de epilepsia [50].

Capítulo 3: Localização de Fontes Neuronais

3.1 Introdução

A interpretação de dados clínicos de EEG passa, na maioria das vezes, pela especulação sobre a localização das fontes contidas no cérebro, responsáveis pela actividade eléctrica observada no escalpe. As primeiras tentativas nesta área começaram há mais de quarenta anos, quando pela primeira vez os investigadores começaram a relacionar os conhecimentos electrofisiológicos do cérebro com as densidades de corrente nos meios condutores [36].

O princípio aplicado consistia em considerar que uma fonte de corrente activa dentro de um meio finito condutor iria produzir uma densidade de corrente que se espalharia pelo meio, criando assim uma diferença de potencial à superfície. Devido à informação sobre como se distribuíam as células piramidais no córtex cerebral (capítulo 1), sugeriu-se que, se algumas dessas células confinadas a determinadas regiões estivessem sincronizadas e fossem paralelas, era possível obter uma densidade de corrente suficiente para produzir uma diferença de potencial no escalpe, que por sua vez poderia ser medida [36].

O processo pelo qual se prevê quais os potenciais no escalpe conhecendo as fontes de corrente contidas no cérebro é designado por problema directo da electroencefalografia [36]. Se todas as configurações e distribuições das fontes, como as propriedades condutoras dos tecidos no interior do volume da cabeça fossem conhecidas em cada instante, poder-se-ia calcular os potenciais em qualquer lugar no escalpe.

Em contra-partida, o processo de prever as localizações das fontes neuronais a partir dos potenciais medidos no escalpe, designa-se por problema inverso [36]. Esta localização é feita com base em métodos matemáticos, que necessitam da criação de modelos para os geradores de correntes e para o meio onde estes se encontram.

3.2 Modelos de Fontes Neuronais

Em primeiro lugar, é necessário definir um modelo para a fonte a ser localizada. Como foi explicado no capítulo 1, as sinapses químicas provocam fluxos iónicos que acontecem do meio extracelular para o meio intracelular e são chamados de corrente transmembranar (entre o meio e a célula). Esta corrente, por sua vez, provoca uma diferença de potencial no interior e exterior do neurónio, causando correntes intra e extracelulares. A corrente intracelular provocada por um estímulo excitatório circula da parte

negativa para a positiva, criando linhas de corrente como no dipolo de corrente. Por este motivo, vão ser apresentados dois modelos, ambos tendo por base o conceito de dipolo.

3.2.1 Modelo de Dipolo Único

O termo dipolo eléctrico pode ser aplicado sempre que existe uma distribuição de correntes com as seguintes características, uma fonte I^+ separada à distância d de um sumidouro I^- com valores absolutos iguais. Esta condição pode ser representada por um fluxo de uma corrente que circula de I^- para I^+. A grandeza momento dipolar $\overrightarrow{m}$ é então definida como sendo um vector com a orientação da recta que une a fonte ao semidouro, representado pela equação 3.1 [42].

$$\overrightarrow{m} = Id\overrightarrow{u}\,. \tag{3.1}$$

A expressão 3.2 representa um potencial V criado por um dipolo de corrente num meio infinito, homogéneo e condutor.

$$V = \frac{I}{4\pi\sigma}\Big(\frac{1}{R_1} - \frac{1}{R_2}\Big), \tag{3.2}$$

onde R_1 é a distância do ponto de medida à fonte, R_2 a distância do ponto de medida ao sumidouro, I é a corrente e σ a condutividade do meio.

Considerando agora um ponto de medida numa posição muito distante $|\overrightarrow{r}| >> |\overrightarrow{d}|$ sendo $\overrightarrow{r}$ o vector que une o ponto de medida ao dipolo e $\overrightarrow{d}$ o vector que une a fonte ao sumidouro, o potencial 3.2 passa a ser o seguinte:

$$V = \frac{Id\cos\vartheta}{4\pi\sigma^2} = \frac{m\cos\vartheta}{4\pi\sigma^2}, \tag{3.3}$$

onde ϑ é o ângulo entre as rectas definidas por $\overrightarrow{d}$ e $\overrightarrow{r}$, e m é o valor absoluto do momento dipolar. O potencial mencionado nas equações 3.2 e 3.3, é calculado para um meio infinito e homogéneo. Se o meio for heterogéneo, o potencial sofre alterações. O potencial será calculado tendo em conta a distribuição de fontes reais e a contribuição de fontes fictícias relacionadas com as heterogeneidades do meio [50].

O modelo apresentado aplica-se a um neurónio, o que não permite obter um sinal mensurável. No entanto, se for considerada uma população de neurónios, com actividade síncrona, com uma disposição paralela entre eles e, no seu conjunto, perpendicular (em relação) ao escalpe, os efeitos das correntes geradas pelos potenciais pós-sinápticos podem ser somados possibilitando assim o seu registo [50].

Desta forma, o modelo do dipolo de corrente pode ser aplicado ao estudo de populações neuronais, tendo em conta a actividade eléctrica conjunta.

3.2.2 Modelo de Múltiplos Dipolos

Existem situações onde o modelo de dipolo único se torna inválido e desajustado, pelo que se torna necessária a utilização de modelos alternativos. Este tipo de modelos é normalmente utilizado quando a população neuronal é demasiado extensa ou quando se regista a presença de várias fontes pontuais com actividades simultâneas. A solução para esta questão poderá passar pela modelação da distribuição de correntes utilizando um conjunto de fontes dipolares (modelos de dipolo único), partindo do princípio que cada um dos geradores neuronais obedece às condições do modelo de dipolo único [50].

3.3 Modelos para a Cabeça

Para procurar a localização de dipolos, é também necessário modelar o meio onde estes se encontram, ou seja, a cabeça. Vão ser apresentados dois modelos e uma variante de um deles: modelo esférico, modelo realista e modelo realista padrão.

3.3.1 Modelo Esférico

A geometria mais utilizada para a modelação da cabeça é a esférica. Assim, o modelo esférico aproxima a cabeça a um conjunto de esferas concêntricas de diferentes condutividades. Nas aplicações mais elaboradas, são utilizadas quatro esferas correspondendo ao: encéfalo, líquido céfalo-raquidiano, crânio e escalpe [50].

3.3.2 Modelo Realista

A superfície da cabeça não é propriamente esférica e a condutividade também não é uniforme para cada estrutura. Estes factos levaram à elaboração de um modelo mais preciso e real.

O modelo realista realiza uma aproximação muito precisa à cabeça e à sua estrutura. No modelo realista, normalmente utiliza-se três superfícies constituídas por pequenos triângulos que formam estruturas geometricamente idênticas às estruturas anatómicas da cabeça [50].

Esta aproximação é feita recorrendo a imagens de Ressonância Magnética Nuclear (RMN) para o paciente em estudo. Este processo torna-se inconveniente e dispendioso devido ao facto de ser necessário que cada paciente realize um exame de RMN à cabeça ou então que o paciente já tenha realizado um.

Para eliminar este problema mantendo uma boa aproximação às estruturas da cabeça, foi desenvolvido o modelo realista padrão que adapta uma superfície da cabeça padrão a cada caso [50]. É um modelo de fácil aplicação que necessita de um crânio de plástico para realizar a aproximação à cabeça [49]. Para tal digitalizaram-se pontos correspondentes ao encéfalo e pontos pertencentes ao crânio (ver figura 3.1). A partir daqui, construiu-se superfícies trianguladas para o encéfalo e para o crânio. Através de uma ampliação da superfície exterior do crânio, conseguiu-se uma terceira superfície referente ao exterior do escalpe. Esta geometria padrão é então adaptada em altura, largura e comprimento ao tamanho externo da cabeça do paciente [49].

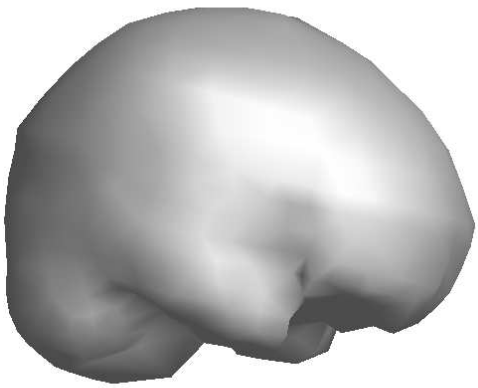


Figura 3.1: Esquema ilustrativo de uma das superfícies utilizadas no modelo realista, neste caso, a superfície do cérebro.

Para os casos que vão ser estudados neste trabalho vão ser utilizados os modelos de dipolo único e de múltiplos dipolos para modelar a fonte neuronal e, para modelar a cabeça, vão ser utilizados o modelo esférico e o modelo realista padrão na sua forma mais simples, ou seja, sem que haja adaptação à cabeça dos indivíduos.

Capítulo 4: Análise em Componentes Independentes

4.1 Introdução

Tal como já se enfatizou nos capítulos anteriores, os sinais EEG obtidos a partir dos eléctrodos resultam da contribuição de um grande número de potenciais produzidos pelas células cerebrais (neurónios). Na análise de PE e EEG, a questão de maior interesse é o conhecimento da localização das fontes cerebrais e para isto seria necessário fazer uma medição directa dos potenciais no interior do encéfalo, ou seja, implicaria recorrer-se a uma cirurgia e colocar vários eléctrodos em determinados pontos no cérebro. Este método, como é óbvio, é indesejável e uma melhor solução será obter a informação desejada a partir dos sinais obtidos nos eléctrodos colocados no escalpe.

Os sinais obtidos nos eléctrodos resultam de somas ponderadas dos potenciais celulares, onde os pesos de cada potencial (sinal) dependem do caminho percorrido entre o neurónio e o eléctrodo (ver figura 4.1).

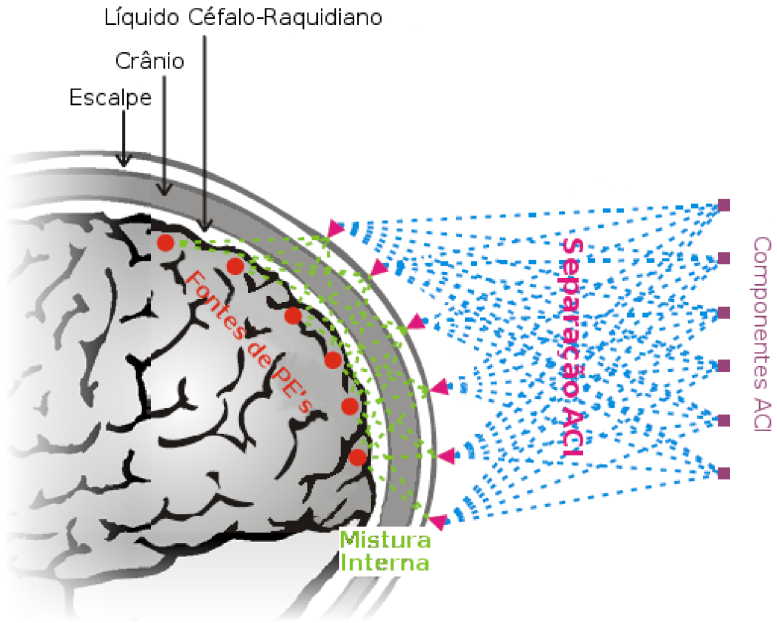


Figura 4.1: Esquema ilustrativo da análise em componentes independentes em EEG (adaptado de [19])

Devido ao facto dos potenciais registados por cada eléctrodo serem o resultado da activação de várias

fontes, os sinais obtidos nos diferentes eléctrodos estão muito correlacionados entre si. Se os coeficientes de ponderação das várias somas fossem conhecidos, seria possível calcular os potenciais nas fontes a partir de um número suficiente de eléctrodos. Neste contexto, a análise em componentes independentes (ACI) constitui uma ferramenta matemática que pode ajudar a resolver este problema através da separação em componentes dos sinais (registados) referentes a um evento.

O exemplo tipo dado para a aplicação da ACI é chamado usualmente de *Coktail-Party Problem* em que a questão consiste numa sala onde existem várias pessoas a falar ao mesmo tempo e o objectivo é identificar o que cada indivíduo está a dizer ouvindo a mistura de vozes. Isto normalmente é feito através da colocação de vários microfones em diversos locais da sala, que irão recolher diferentes misturas dos sinais de voz.

4.2 Definição da ACI

A ACI é um método estatístico, computacional, que tem como objectivo revelar fontes escondidas que originam conjuntos de medidas ou sinais (em geral, obtidos através de sensores). Basicamente, a ACI faz a separação de sinais multivariados (sinais resultantes de uma mistura de várias fontes) em componentes, que são consideradas como aproximações das fontes.

Olhando para um exemplo matemático simples, tendo duas fontes e dois sensores, pode-se dizer que $x_1(t)$ e $x_2(t)$ são combinações lineares das fontes $s_1(t)$ e $s_2(t)$ expressas nas equações 4.1 e 4.2.

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) \tag{4.1}$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t). \tag{4.2}$$

O objectivo da ACI é achar a_{11}, a_{12}, a_{21} e a_{22}. Tendo estes coeficientes é possível fazer estimações de $s_1(t)$ e $s_2(t)$. Para a resolução do sistema formado pelas equações 4.1 e 4.2 é usada uma notação matricial passando este a ser representado pela equação 4.3:

$$X = A\,S, \tag{4.3}$$

denominando-se A a matriz de mistura, X o vector dos sinais medidos nos sensores e S o vector das fontes. O principal objectivo dos algoritmos de ACI é o cálculo desta matriz A (o que na prática passa pelo cálculo da matriz de separação W cuja a inversa é a matriz de mistura A). A multiplicação da matriz W por x origina as componentes y que são uma estimação das fontes originais.

$$Y = W X. \tag{4.4}$$

Em seguida, é feita uma demonstração gráfica do funcionamento da ACI usando um dos algoritmos estudos, o Jade. Na primeira fila da figura 4.2 são mostradas as fontes utilizadas: sinusoidal, quadrada e ruído Gaussiano. Estas fontes foram multiplicadas pela seguinte matriz de mistura A:

$$\mathbf{A} = \begin{bmatrix} 2.2475 & -0.3846 & 0.7345 \\ -0.2164 & -0.1896 & 0.6205 \\ 1.0196 & -0.6836 & 1.3621 \end{bmatrix}$$

O resultado dessa multiplicação é dado na segunda fila da figura 4.2, ou seja, são obtidas as misturas. O algoritmo de ACI é então aplicado a essas misturas.

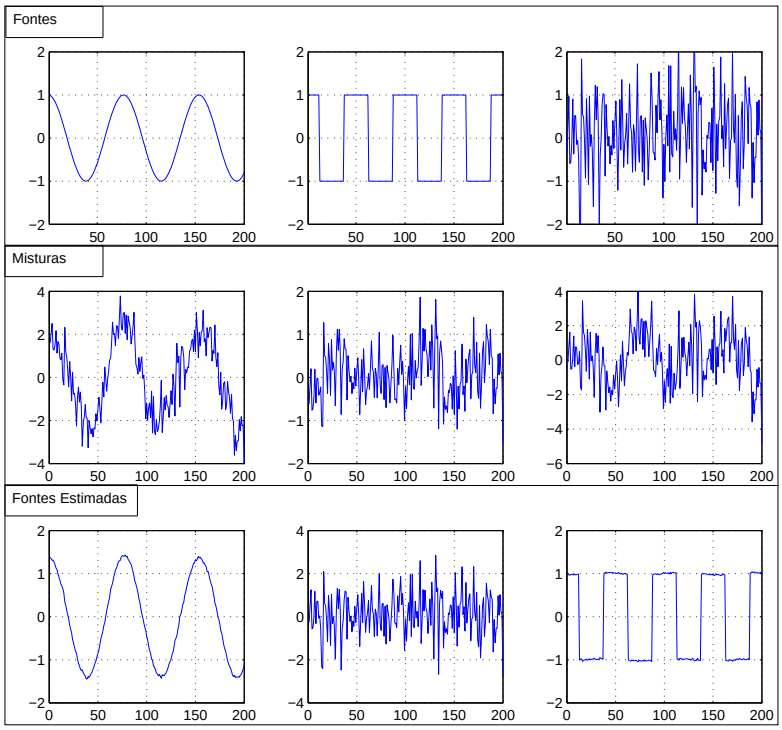


Figura 4.2: Exemplo da aplicação da ACI.

Na terceira fila da figura 4.2 estão representadas as componentes calculadas pela ACI, que são estimações da fontes originais. Como se pode verificar, embora a forma das fontes em termos de comportamento temporal tenha sido recuperada, o mesmo não aconteceu ao nível das suas amplitudes (uma das limitações da ACI).

4.3 Características da Análise em Componentes Independentes

Os algoritmos de ACI estudados para este trabalho têm as seguintes características/condicionantes [29]:

1. As fontes são consideradas estatisticamente independentes.
2. Apenas uma das fontes pode ter uma distribuição Gaussiana, devendo as outras ser ou sub-Gaussianas ou super-Gaussianas.
3. É assumido que cada sinal de cada sensor representa uma diferente combinação linear das fontes.
4. Não existem atrasos temporais entre as fontes e os sensores.
5. No máximo, são identificáveis um número de componentes igual ao número de sensores.
6. A energia das fontes não poder ser determinada.
7. O sinal (positivo ou negativo) das fontes não poderá ser determinado.
8. A ACI encontra sempre um sub-espaço onde uma determinada componente é independente das restantes encontradas.

A independência estatística pode ser definida considerando duas variáveis aleatórias s_1 e s_2. Basicamente, as variáveis s_1 e s_2 são consideradas independentes se a informação do valor de s_1 não der nenhuma informação do valor de s_2 e vice-versa [29]. Tecnicamente, a independência pode ser definida através de densidades de probabilidade. Denotando $p(s_1, s_2)$ como a função densidade de probabilidade conjunta das variáveis s_1 e s_2, e denotando $p_1(s_1)$ como a função densidade de probabilidade marginal de s_1, isto é, a função densidade de probabilidade de s_1 se esta for considerada isoladamente. As funções densidade de probabilidade $p_1(s_1)$ e $p_2(s_2)$ são dadas pelas expressões 4.5 e 4.6, respectivamente.

$$p_1(s_1) = \int p(s_1, s_2) ds_2 \tag{4.5}$$

$$p_2(s_2) = \int p(s_1, s_2) ds_1. \tag{4.6}$$

Então define-se que s_1 e s_2 são independentes se e só se a função densidade de probabilidade conjunta $p(s_1, s_2)$ for factorizável, ou seja, se a função densidade de probabilidade conjunta das duas variáveis for igual à multiplicação das funções densidade de probabilidade marginal de cada uma das variáveis [29]:

$$p(s_1, s_2) = p_1(s_1)p_2(s_2), \tag{4.7}$$

Esta definição estende-se naturalmente a n variáveis aleatórias.

Outra das condicionantes é a de só poder haver, no máximo, uma fonte com distribuição Gaussiana, ou seja, a impossibilidade de separação de fontes Gaussianas. Esta situação explica-se da seguinte forma: assumindo que a matriz de mistura é ortogonal e que duas fontes denotadas como s_1 e s_2 são Gaussianas, as misturas x_1 e x_2 vão ser Gaussianas, não correlacionadas e com variância unitária. A função densidade de probabilidade conjunta é dada pela equação 4.8 [29]:

$$p(x_1, x_2) = \frac{1}{2\pi} exp(-\frac{x_1^2 + x_2^2}{2}). \tag{4.8}$$

Esta distribuição encontra-se ilustrada na figura 4.3.

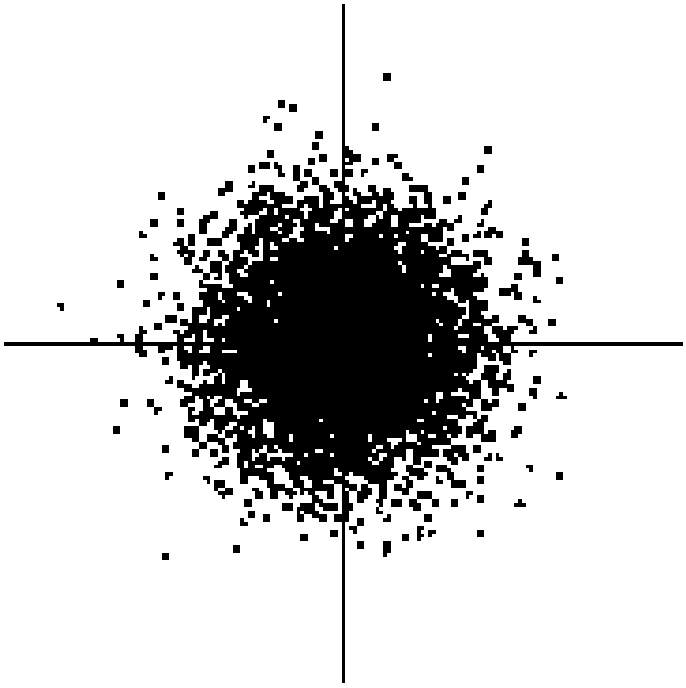


Figura 4.3: Função densidade de probabilidade conjunta de duas misturas Gaussianas [29].

A figura 4.3 mostra que a densidade é completamente simétrica, assim, esta não contém nenhuma informação sobre as direcções das colunas da matriz de mistura. Desta forma a matriz de mistura não pode ser estimada [29].

4.3.1 Medidas de Independência para Separação das Componentes

4.3.1.1 Maximização da Não-Gaussianidade

Como não é possível estimar fontes Gaussianas independentes, torna-se lógico que um dos métodos para a estimação de fontes passe pela procura de componentes cuja distribuição não seja Gaussiana, ou seja, tenham uma distribuição super ou sub-Gaussiana. Segundo o Teorema do Limite Central, em determinadas condições, a distribuição de uma soma de variáveis aleatórias independentes tende a ser uma distribuição Gaussiana, ou seja, a distribuição da soma das variáveis é mais Gaussiana que qualquer uma das distribuições das variáveis [29]. Como a ACI não consegue separar fontes Gaussianas, o processo de estimação da matriz de mistura utilizado pela ACI passa pelo cálculo dos vectores da matriz de separação (matriz de mistura inversa) que irão maximizar a não-Gaussianidade das componentes (fontes estimadas), resultantes da multiplicação desta com os dados originais.

Para a maximização da não-Gaussianidade, é necessário uma medida que permita quantificar a Gaussianidade de uma componente. Existem vários métodos para fazer isto, entre os quais a análise da *Kurtosis* utilizada no algoritmo Jade por exemplo, e a *negentropia* (entropia negativa) utilizada no algoritmo Fastica. A *Kurtosis* é um cumulante (anexo E) de quarta ordem e a *Kurtosis* de uma variável y é dada pela expressão 4.9 [29]:

$$kurt(y) = E\{y^4\} - 3(E\{y^2\})^2 \tag{4.9}$$

em que $kurt(y)$ é:

- Igual a 0 se y tiver uma distribuição Gaussiana.
- Maior que 0 se y tiver uma distribuição super-Gaussiana.
- Menor que 0 se y tiver uma distribuição sub-Gaussiana.

4.3.1.2 Minimização da Informação Mútua

Existe outro método para a separação das componentes que é a minimização da informação mútua dos sinais observados.

Definindo um vector $s(t) = [s_1(t), s_2(t), \ldots, s_N(t)]^T$, de forma que s_i constitua uma sequência finita de variáveis estocásticas com média zero e mutuamente independentes, então $s(t)$ resulta da contribuição de N fontes independentes. Como já foi referido, devido à independência das fontes, a função densidade de probabilidade conjunta de s é o produto de N distribuições marginais $p_i(s_i)$:

$$p(s) = \prod_{i=1}^{N} p_i(s_i). \tag{4.10}$$

Um igual número N de misturas pode ser derivado de acordo com o modelo linear dado pela expressão 4.3. O resultado é um vector $x(t) = [x_1(t), x_2(t), \ldots, x_N(t)]^T$. Em ACI, a informação mútua é definida pela divergência da densidade multivariada (conjunta) de Kullback-Leibler que é dada pela expressão 4.11 [52][17].

$$I(x) = \int p(x) \log \frac{p(x)}{\prod_{i=1}^{N} p_i(x_i)} dx. \tag{4.11}$$

A informação mútua $I(x)$ é sempre positiva, sendo 0 no caso das componentes x serem independentes [52][29]. O algoritmo Infomax (IcaML) utiliza a maximização da log-verosimilhança que é equivalente à minimização da informação mútua [29][17].

Estas medidas de independência estão explicadas com maior detalhe nos trabalhos de Pierre Comon [17] e de Aapo Hyvärinen [29].

4.4 Sistema Representativo do Funcionamento da ACI

Os algoritmos estudados para este trabalho foram: o Fastica, o Infomax(IcaML) e o Jade. O sistema que define o funcionamento destes algoritmos pode ser claramente dividido em três fases: as fases de pré-processamento, processamento e pós-processamento. Este sistema encontra-se esquematizado na figura 4.4.

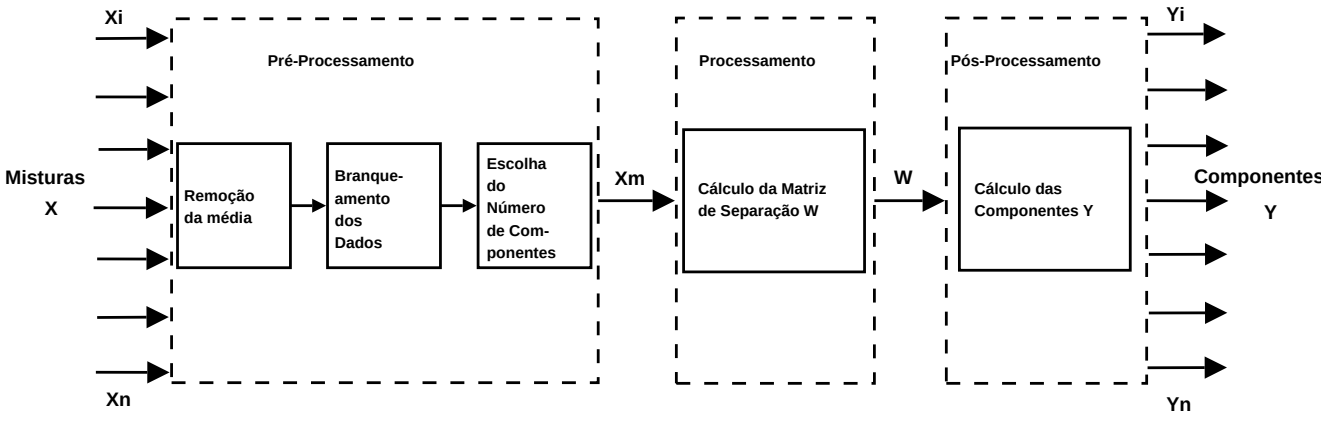


Figura 4.4: Esquema de funcionamento de um algoritmo de ACI.

A fase de pré-processamento é constituída pelas seguinte etapas:

1. Remoção da média dos sinais misturados denotados pela matriz X (centrar sinais).
2. Branqueamento dos dados (calcular matriz de branqueamento).
3. Análise em componentes Principais (ACP), se for necessário a escolha do número de componentes a ser calculado.

À saída desta fase obtemos uma matriz denotada X_m que representa os dados centrados, branqueados, e, se foi escolhido o número de componentes a ser calculado, reduzidos.

A fase de processamento corresponde ao cálculo da matriz de separação W a partir da matriz X_m.

A fase de pós-processamento constitui a multiplicação da matriz de separação W com os sinais misturados originais X para se obter as fontes estimadas (componentes). A matriz de mistura A pode também ser obtida através de $A = W^{-1}$.

4.5 Pré-Processamento

4.5.1 Centrar Sinais

Em primeiro lugar, os sinais misturados precisam de ser centrados (média zero). Isto é feito através da subtracção da média de cada sinal misturado,

$$X_c = X - E\{X\}, \tag{4.12}$$

onde E simboliza a esperança matemática.

4.5.2 Branqueamento dos dados

Depois de centrados os dados, uma das operações mais úteis no pré-processamento é branquear os dados observados (variáveis observadas). Isto significa transformar X (depois de centrado) através de processos lineares de forma a obter uma nova matriz X branqueada, ou seja, os valores entre sinais estão descorrelados e a variância entre sinais é unitária o que equivale a dizer que a matriz de covariância de X iguala a matriz identidade:

$$COV(X_c) = E\{X_c X_c^T\} = I, \tag{4.13}$$

Um dos métodos mais usados para a realização desta operação é a decomposição dos valores próprios da matriz $E\{X_c X_c^T\}$. Esta decomposição é dada por:

$$E\{X_c X_c^T\} = VDV^T, \tag{4.14}$$

onde V é uma matriz ortogonal dos vectores próprios de $E\{X_c X_c^T\}$ e D é uma matriz diagonal com os valores próprios de $E\{X_c X_c^T\}$, $D = diag(d_1, \ldots, d_n)$. O branqueamento é feito através de:

$$X_m = VD^{-1/2}V^T X_c, \tag{4.15}$$

em que $D^{-1/2} = diag(d_1^{-1/2}, \ldots, d_n^{-1/2})$. Por 4.15 retira-se que a matriz de branqueamento é $W_h = VD^{-1/2}V^T$.

Pode ser também útil reduzir as dimensões dos dados ao mesmo tempo que são branqueados. Isto é feito através da escolha dos valores próprios d_j de $E\{X_c X_c^T\}$ de modo a ignorar os valores próprios nulos ou aqueles que se encontram abaixo de um valor pré-definido. Ao ser feita esta escolha, os vectores próprios da matriz $E\{X_c X_c^T\}$ com maior energia são conservados, mantendo-se assim os sub-espaços de maior relevância no conjunto dos dados observados. Esta operação é chamada de análise em componentes principais (ACP) e permite a redução do ruído e do *overlearning*. O problema do *overlearning* será tratado no próximo capítulo.

4.6 Processamento

Como já foi referido, a fase de processamento passa, principalmente, pelo cálculo da matriz de separação W (ou a matriz de mistura). Em seguida, vão ser apresentados os métodos utilizados por cada um dos algoritmos estudados para fazer o cálculo da matriz W. Para cada um serão enunciados não só o tipo de fontes que são capazes de separar, como também as suas vantagens e desvantagens.

4.6.1 Fastica

O algoritmo Fastica utiliza um método de ponto fixo que usa como medida da não-Gaussianidade uma aproximação à negentropia (entropia negativa), que neste caso é uma aproximação baseada no princípio da máxima entropia [29]. As vantagens do algoritmo FastIca são [29]:

- A convergência é cúbica, o que torna o algoritmo muito eficiente.

- Consegue separar sub e super-Gaussianas, dependendo dos parâmetros utilizados.
- Na versão de cálculo das componentes por deflação, existe uma poupança nos recursos de memória, o que permite calcular um grande número de componentes.
- Permite a escolha do tipo de distribuição das fontes que pretendemos separar.
- Permite privilegiar um determinado tipo de distribuição de fonte.

As desvantagens são [29]:

- Exigir um grande número de parâmetros para adaptação ao problema a resolver (estes parâmetros estão enunciados no anexo A).
- Apresentar problemas de convergência quando não são escolhidos os parâmetros adequados.

Este algoritmo é explicado com maior pormenor no anexo A deste relatório. O software Matlab que implementa este algoritmo encontra-se disponível em [3].

4.6.2 Infomax (IcaML)

O algoritmo Infomax utiliza a maximização da log-verosimilhança (minimização da informação mútua) para separar as componentes independentes [52]. No caso do algoritmo IcaML, a maximização da log-verosimilhança é feita de uma forma mais precisa através da utilização do algoritmo UCMINF (algoritmo tipo BFGS (Broyden-Fletcher-Goldfarb-Shanno)) [35] que utiliza optimizações não-lineares para o cálculo de mínimos locais [41]. O algoritmo Infomax (IcaML) tem como principais vantagens [35]:

- A possibilidade de escolha do tipo de distribuição das fontes que pretendemos separar (no caso do algoritmo Infomax original).
- O cálculo imediato da log-verosimilhança (logaritmo da densidade de probabilidade) da ACI para o número de componentes utilizados.

A principal desvantagem deste algoritmo é a de ter uma maior complexidade, o que diminui a sua velocidade.

Este algoritmo é explicado com maior pormenor no anexo B deste relatório. O software Matlab utilizado que implementa este algoritmo encontra-se disponível em [4].

4.6.3 Jade

O algoritmo Jade efectua a ACI através da diagonalização conjunta da matriz de todos os cumulantes de segunda e quarta ordem dos dados a separar [14]. Este algoritmo, ao contrário dos outros aqui apresentados, não faz nenhuma estimação ou assunção da distribuição das fontes, e, devido a isto, consegue separar componentes sub e super-Gaussianas simultaneamente [14].

Uma das vantagens da utilização dos cumulantes é que o algoritmo é totalmente algébrico, apresentando características sequenciais, o que elimina problemas de convergência. Uma das consequências desta vantagem é a de que este algoritmo não necessita de parâmetros [38][15].

As desvantagens deste algoritmo são a exigência de grandes recursos de memória devido à necessidade de armazenamento das matrizes de cumulantes cruzados de segunda e quarta ordem, e da impossibilidade de escolha da distribuição das componentes a separar.

Este algoritmo é explicado com maior pormenor no anexo C deste relatório. O software Matlab utilizado que implementa este algoritmo encontra-se disponível em [1].

4.7 Pós-Processamento

À saída da fase de processamento obtém-se a matriz de separação W. As componentes independentes são obtidas através da equação 4.16.

$$Y = W\,X, \tag{4.16}$$

onde Y é a matriz de componentes e X a matriz de sinais medidos. Depois de escolhidas as componentes de interesse, é necessário fazer a projecção da componente sobre os eléctrodos (escalpe). Essas projecções são obtidas através da equação 4.17.

$$X_{proj} = [W^{-1}]_n\, y_n, \tag{4.17}$$

onde X_{proj} são as projecções sobre os eléctrodos, y_n é uma das componentes, $[W^{-1}]_n$ é o vector de mistura da componente n (A_n) obtida através do cálculo da matriz inversa da matriz de separação W. No caso de redução do número de componentes, em vez da inversa é calculada a pseudo-inversa da matriz de separação W.

A pseudo-inversa permite calcular a inversa de uma matriz não quadrada. Normalmente é utilizada a pseudo-inversa de *Moore-Penrose* [34]. A pseudo-inversa U de uma matriz V tem as mesmas dimensões da matriz V^T e deve ter as seguintes características:

- $V\,U\,V = V$
- $U\,V\,U = U$
- $V\,U$ é Hermitiana
- $U\,V$ é Hermitiana

O cálculo da pseudo-inversa é baseado na aplicação da técnica de decomposição em valores singulares (SVD) [34].

4.8 Escolha do Algoritmo

Para a aplicação da ACI na análise de PE, vai ser escolhido apenas um dos algoritmos anteriormente apresentados. Para escolher um deles é necessário levar em conta algumas características dos potenciais evocados.

Normalmente, assume-se que os PE são compostos por uma ou mais breves activações sobrepostas em que cada uma corresponde à activação de uma determinada área cerebral, a qual está envolvida numa ou mais etapas do processamento do estímulo. Essas breves activações caracterizam-se normalmente por apresentarem distribuições super-gaussianas [38].

Os testes apresentados neste sub-capítulo foram efectuados em ambiente MATLAB utilizando os algoritmos de ACI anteriormente referidos, recorrendo às implementações já existentes destes algoritmos que estão disponíveis nos sítios anteriormente referidos.

Foi criado um conjunto de sinais com distribuições super-gaussianas. As super-gaussianas foram geradas a partir do seno hiperbólico, $\sinh(x)$, de uma variável aleatória com distribuição normal. O sinal obtido é depois normalizado para que este tenha média igual a zero e variância igual a um.

Como os dados fornecidos para os casos reais em estudo são constituídos por 21 sinais de 256 pontos, o conjunto de sinais vai ter também o mesmo formato. Estes sinais representam as fontes que queremos recuperar. Estas fontes foram então misturadas através de uma matriz de mistura A aleatória. Multiplicando a matriz A pelas fontes, obtêm-se os sinais misturados (simulando os dados obtidos nos sensores)

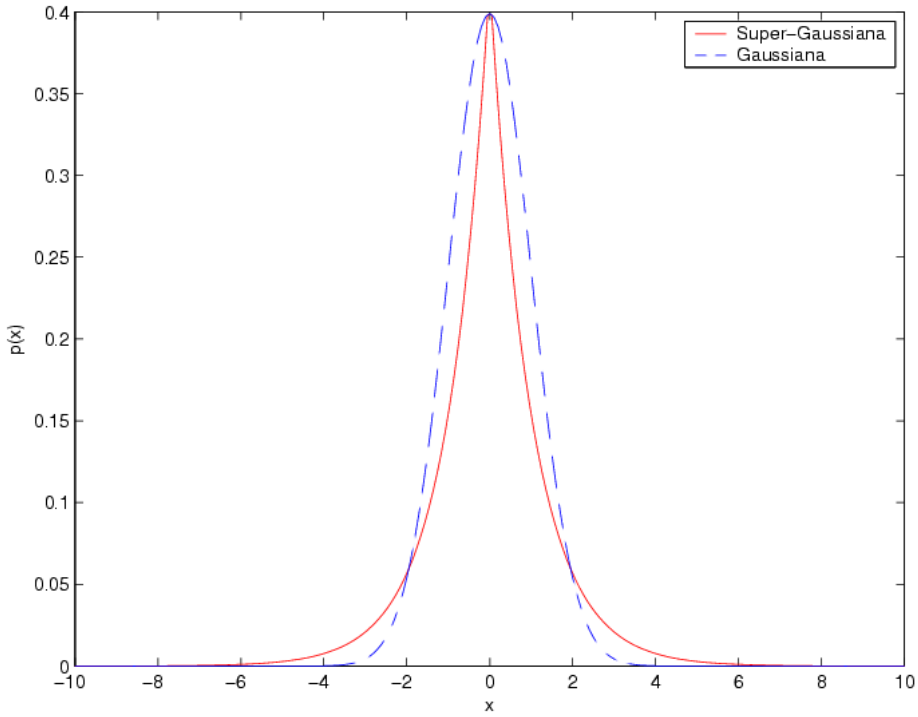


Figura 4.5: Distribuições Gaussianas e Super-Gaussianas

para poder avaliar o desempenho de cada algoritmo no processo de separação de distribuições super-gaussianas. Foi feita também a simulação de nove ambientes ruidosos (adicionando-se às misturas ruído branco gaussiano com diferentes relações sinal-ruído) e um ambiente sem ruído. Para cada um destes ambientes foram criadas 20 matrizes de mistura aleatórias criando-se assim 20 conjuntos de misturas. Os algoritmos foram aplicados a cada conjunto e o seu desempenho foi avaliado através do uso do índice SIR [54] (*Signal Interference Ratio*) que é definido pela equação 4.18:

$$SIR = \frac{1}{n}\sum_{i,j=1}^{n}\frac{|[WA]_{ij}|}{max_{1\leq k\leq n}(|[WA]_{ik}|)}, \tag{4.18}$$

em que A é a matriz de mistura e W a matriz de separação estimada pelos algoritmos. De notar que quanto menor for o valor do SIR melhor é o desempenho do algoritmo analisado. Se o valor do SIR for zero então temos uma recuperação perfeita (querendo dizer que a matriz de permutação $P = WA$ é igual à matriz identidade I). Os resultados obtidos são mostrados na tabela 4.1.

SNR(dB)	Jade	IcaML	FastIca
0dB	0.4371	0.4699	0.4511
5dB	0.3474	0.3418	0.3523
7dB	0.3199	0.2734	0.2798
10dB	0.2830	0.2259	0.2273
13dB	0.2626	0.2050	0.2053
15dB	0.2338	0.1680	0.1758
17dB	0.2271	0.1492	0.1571
20dB	0.2273	0.1407	0.1470
Sem Ruído	0.1807	0.0866	0.1007

Tabela 4.1: Resultados médios do SIR para cada um dos algoritmos e para cada relação sinal ruído.

Observando a tabela verifica-se que o algoritmo que apresenta melhores resultados globais para o caso em estudo será o algoritmo IcaML. O algoritmo IcaML foi especialmente desenvolvido para separar unicamente super-Gaussianas. O FastIca apresenta também bons resultados utilizando os seguintes parâmetros: a função de não-linearidade utilizada foi a $tanh(x)$, o método de procura das componentes foi o método simétrico e a condição de paragem do algoritmo foi a da realização de 5000 iterações, para evitar a não convergência do algoritmo.

Capítulo 5: Estimação do Número de Componentes

5.1 Introdução

No capítulo anterior foi introduzido o problema do *overlearning* que acontece quando a ACI encontra uma componente que não é independente de outra componente já encontrada, mas no entanto, a ACI encontra sempre um sub-espaço onde ela é independente. Esta situação acontece quando o número de misturas é maior que o número de fontes.

A situação de *overlearning* nos algoritmos de ACI conduz a uma separação completamente errada das fontes presentes nas misturas. Esta situação é de difícil identificação devido ao desconhecimento, em muitos casos, do número de fontes presentes nas misturas e devido ao facto de que a matriz de separação encontrada pelo algoritmo de ACI ajustar-se perfeitamente às misturas, levando assim ao cálculo de componentes estatisticamente independentes, mas inválidas para o problema em estudo, como vai ser demonstrado no exemplo gráfico apresentado na próxima secção.

O problema de *overlearning* em potenciais evocados foi verificado em [21], podendo ocorrer muitas vezes em situações onde se utilizam um elevado número de eléctrodos, obtendo-se um maior número de misturas.

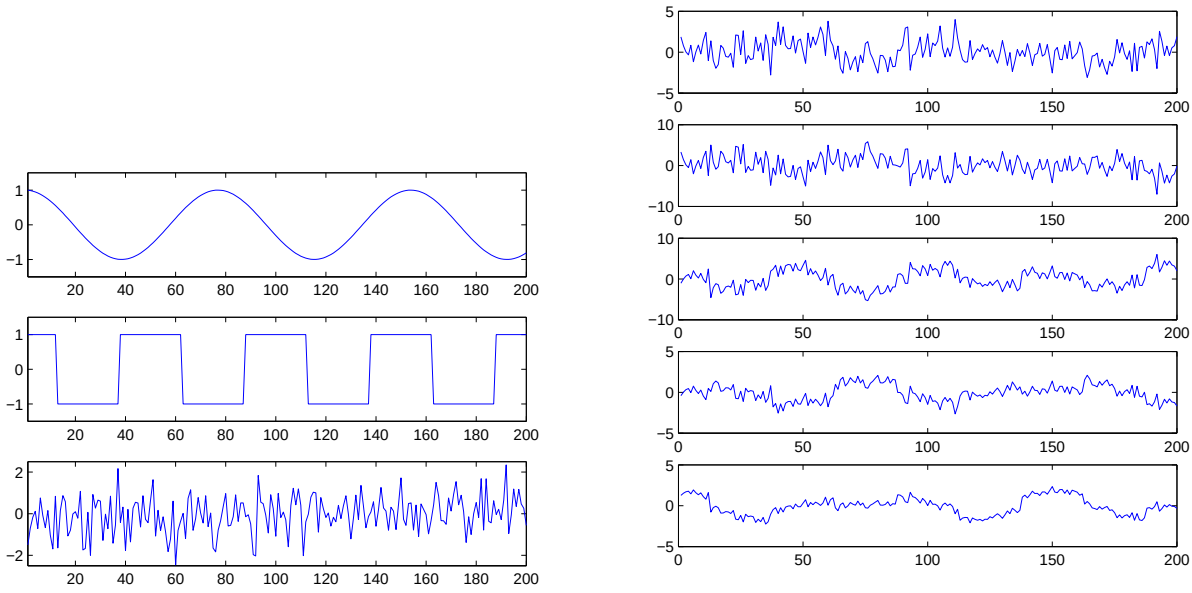
Uma solução passa então pela redução de componentes. É sobre esta solução que este trabalho se debruça, a tentativa de redução do número de dimensões para evitar a sub-divisão de componentes que pertençam ao mesmo evento, mais precisamente no estudo dos potenciais $P300$.

5.2 Demonstração Gráfica do Problema do *Overlearning*

Segue-se uma demonstração do problema do *overlearning* através de um exemplo gráfico. Três sinais (fontes): um sinal sinusoidal, um sinal quadrado e um sinal ruidoso, estão apresentados na figura 5.1(a). Estes sinais foram misturados utilizando a seguinte matriz de mistura (simulando o meio) cujos valores, entre 1 e -1, foram obtidos aleatoriamente:

$$\mathbf{A} = \begin{bmatrix} 0.7293 & 0.5228 & -1.3833 \\ 0.6945 & -0.8721 & -2.3240 \\ -1.0215 & 1.9223 & 1.3068 \\ 0.9993 & -0.5522 & 0.5739 \\ 1.0682 & 0.7272 & 0.3487 \end{bmatrix}$$

Os sinais misturados são obtidos através da multiplicação das fontes pela matriz *A* (mistura linear). Os sinais obtidos a partir dessa multiplicação são apresentados na figura 5.1(b).



(a) Sinais originais (fontes).

(b) Sinais misturados.

Figura 5.1: a) Sinais originais (sinal sinusoidal, quadrado e de ruído branco). b) Sinais resultantes da mistura das fontes apresentadas na figura 5.1(a).

Os sinais da figura 5.1(b) são os sinais que são medidos nos sensores (cinco sensores neste caso). Aplicando agora a ACI a estes sinais, utilizando o procedimento habitual que é o de encontrar um número de componentes igual ao número de misturas (sensores), obtêm-se as componentes ilustradas na figura 5.2.

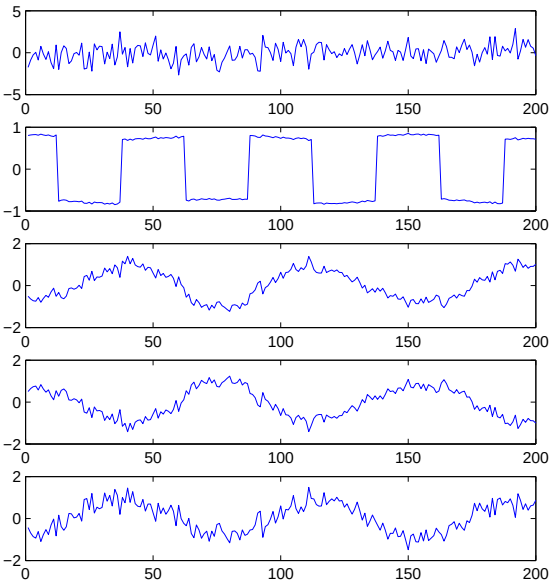


Figura 5.2: Fontes recuperadas através da ACI.

Verifica-se na figura 5.2 que a fonte sinusoidal foi sub-dividida em três componentes. Tanto estas componentes, como a componente quadrada apresentam algum ruído. Aplicando agora a ACI às mesmas misturas mas reduzindo o número de componentes a procurar (para um número igual ao número de fontes), obtêm-se as componentes ilustradas na figura 5.3.

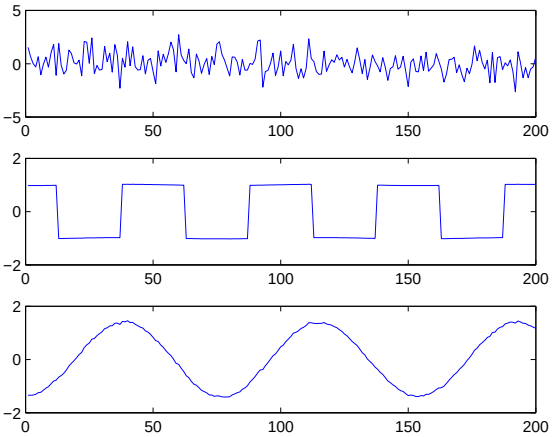


Figura 5.3: Fontes recuperadas através da ACI, mas reduzindo o número de componentes a encontrar.

Neste caso já não houve sub-divisão das componentes e o ruído das componentes foi reduzido. A redução do número de fontes a separar levou a uma recuperação (estimação) das fontes originais. Ficou, pois, completamente provada a necessidade de proceder à redução do número de componentes a procurar, quando de pretende uma correcta recuperação das fontes.

5.3 Estimação do Número de Componentes

Como acabou de ser demonstrado, a solução para a redução dos artefactos e do ruído e, principalmente, para evitar a sub-divisão das componentes de interesse a fim de facilitar a identificação destas, passa pela redução do número de componentes a calcular, ou seja, reduzir o número de sub-espaços que se aceita estarem presentes no conjunto de dados. A redução do número de componentes traz um novo problema: até que ponto se pode reduzir o número de sub-espaços sem eliminar dados que podem ser importantes para o nosso estudo. O ideal seria saber o número de fontes que deram origem às misturas que foram medidas. Como já foi referido, na prática, isto não é possível. Existem, no entanto, ferramentas capazes de fazer uma estimação do número provável de fontes presentes num conjunto de misturas.

Para este trabalho foram consideradas três ferramentas: o Akaike Information Criterion (AIC) [26], o Bayesian Information Criterion (BIC) [22] e o Minimum Description Length (MDL) [31].

5.3.1 Bayesian Information Criterion

Considerando um conjunto de modelos $m = 0, \ldots, M$, pretende-se maximizar a probabilidade de um modelo m, para um conjunto de dados observados X. Usando a relação de Bayes, esta probabilidade pode ser escrita através da expressão 5.1 [35]:

$$p(m|X) = \frac{p(X|m)p(m)}{p(X)}, \tag{5.1}$$

onde $p(m)$ é a probabilidade do modelo. Um modelo é normalmente definido por um conjunto de parâmetros θ. Desta forma obtém-se a probabilidade $p(X|\theta, m)$ e a relação 5.2 é calculada [35]:

$$p(X|m) = \int p(X, \theta|m)d\theta = \int p(X|\theta, m)p(\theta|m)d\theta, \tag{5.2}$$

onde $p(\theta|m)$ representa a probabilidade dos parâmetros para um determinado modelo, que muitas vezes é assumido não ter grande influência no integral 5.2. O integral da equação 5.2 é muito complicado para ser resolvido de uma forma analítica [35]. Foram sugeridos vários esquemas de aproximação a este integral. O método BIC (Bayesian Information Criterion) faz a seguinte aproximação ao integral [22][35]:

$$p(X|m) \approx p(X|\theta, m)p(\theta, m)(N)^{-\frac{d}{2}}, \tag{5.3}$$

em que d é a dimensão do vector de parâmetros θ, e N é o número de amostras. Como o número de parâmetros do modelo é conhecido então $p(\theta, m) = 1$ [35].

O critério BIC é baseado no indicador *bic* dado pela expressão 5.4 [22]:

$$bic = -log(p(X|\theta, m)) + \frac{d}{2}log(N). \tag{5.4}$$

Na aplicação deste critério ao caso da ACI, denota-se modelo pelo número de componentes que se pretende (sub-espaços), sendo $m = K$ onde K é o número de componentes a separar. Quanto menor for o valor do *bic* maior será probabilidade de um determinado modelo m ACI (com um número de componentes K e conjunto de parâmetros θ) se adequar aos dados observados X. O critério BIC aplicado aos modelos de ACI encontra-se melhor explicado no anexo D.

5.3.2 Minimum Description Length

O método MDL trata os modelos como ferramentas de codificação de dados e favorece os que são mais eficientes. Um conjunto de dados pode ser inteiramente representado através de um conjunto de parâmetros seguido de uma "string" comprimida desses dados, que podem ser reconstruídos com referência aos parâmetros representativos calculados [25]. O critério MDL selecciona o modelo que melhor poderá ser representado desta da forma [25]. Existem muitas maneiras para calcular o MDL, mas, no caso do número máximo de parâmetros ser conhecido e fixo e de N ser grande, Rissanen deduziu a seguinte expressão para o cálculo do MDL [31][25]:

$$mdl = -log(p(X|\theta, m)) + \frac{d}{2}log(N). \tag{5.5}$$

Como se pode verificar, para este caso, o critério MDL é equivalente ao critério BIC apresentado por Schwarz [25][20].

5.3.3 Akaike Information Criterion

O critério AIC faz uma aproximação assimptótica à entropia do modelo, ao contrário do método BIC que faz uma aproximação à probabilidade posterior [18][43]. Este método não foi testado neste trabalho por apresentar piores desempenhos que os restantes métodos estudados, como se pode verificar nos ensaios realizados em [18] e em [43].

5.4 Desempenho do Critério BIC/MDL

Devido à equivalência entre os critérios BIC e MDL para este caso e o melhor desempenho revelado por estes dois em relação ao critério AIC, o método escolhido para este estudo foi o BIC/MDL. Sendo assim, o critério BIC/MDL foi avaliado na detecção de um número diferente de fontes super-Gaussianas presentes em 21 misturas aleatórias em diferentes ambientes de ruído.

Os testes foram efectuados em ambiente MATLAB utilizando o algoritmo IcaML para a ACI, este algoritmo revelou-se o melhor a separar fontes super-Gaussianas nos testes realizados. O software Matlab que implementa o critério BIC encontra-se disponível em [4].

Mais uma vez, as super-gaussianas foram geradas a partir do seno hiperbólico, sinh(x), de uma variável aleatória com distribuição normal. O sinal obtido é depois normalizado para que este tenha média igual a zero e variância igual a um. Para cada um dos casos foram utilizados 20 conjuntos de sinais diferentes para o cálculo das percentagens. Os resultados obtidos são mostrados na tabela 5.1.

Fontes\SNR	**0dB**	**5dB**	**7dB**	**10dB**	**15dB**	**20dB**	**Sem Ruído**
3	100%	100%	100%	100%	100%	100%	100%
5	95%	100%	100%	100%	100%	100%	100%
10	0%	0%	45%	90%	100%	100%	100%
15	0%	0%	0%	0%	0%	95%	100%
21	0%	0%	0%	0%	0%	0%	100%

Tabela 5.1: Percentagem de acerto do método BIC para cada ambiente de ruído e para cada número de fontes

Como se pode verificar, tanto o número de fontes presentes, como o nível de ruído afectam muito a detecção das fontes presentes na misturas. O critério BIC/MDL apresenta principalmente uma grande sensibilidade ao nível de ruído.

Capítulo 6: Análise Experimental

6.1 Dados Utilizados

Os dados utilizados neste trabalho são referentes a potenciais electroencefalográficos cognitivos, *P*300 auditivos (estímulos auditivos). Este exame foi dividido em duas partes: na primeira sujeitou-se os indivíduos a uma série de estímulos facilmente identificáveis; por exemplo um som com uma dada frequência a que chamaremos *Bip*. Este estímulo gerará uma resposta cerebral padrão. Numa segunda fase, sujeitou-se os indivíduos a dois sons distintos intercalados: o frequente, com características iguais às do primeiro estímulo (*Bip*) e um outro, menos frequente, que aparece aleatoriamente e tem, relativamente ao primeiro, uma frequência diferente (*Bop*). A tarefa do paciente foi detectar e contar os estímulos menos frequentes (*Bop*) [23], que é o paradigma utilizado no serviço de Neurofisiologia do IPOFG de Lisboa, onde os dados foram recolhidos e com o qual se obteve os registos explorados nesta tese.

Neste trabalho vão ser utilizados dados referentes a indivíduos normais e a indivíduos com epilepsia temporal complexa. Destes conjuntos, seleccionaram-se quatro indivíduos normais e quatro doentes, seguindo um critério baseado na latência do *P*300 e do *N*100, ou seja, escolheram-se os indivíduos cujos sinais têm amplitudes bem definidas nas zonas dos 300ms e 100ms, respectivamente. Cada registo prolonga-se por 512ms. Os dados estão em formato de texto, numa matriz de 768×7 que corresponde a 256 amostras referentes a cada um dos 21 eléctrodos utilizados.

6.2 Procedimentos

O método de visualização dos potenciais registados pelos vários eléctrodos para a identificação dos intervalos (latências) onde vão ser escolhidas as componentes independentes é o método do *envelope*. Este método passa pela representação gráfica de dois sinais artificialmente gerados: um é constituído pelo valor máximo medido nos eléctrodos em cada instante e o outro é constituído pelo seu valor mínimo.

O algoritmo de ACI foi, em primeiro lugar, aplicado na busca do máximo número de componentes, ou seja, neste caso, 21. Depois foram seleccionadas as componentes cujos picos se encontravam nas latências escolhidas anteriormente para o *P*300 e para o *N*100. Este passo foi sempre efectuado para verificar se haveria perda de alguma componente relevante e verificar a existência de *overlearning*, embora no presente trabalho, se apresente apenas estes resultados para o primeiro caso analisado.

Seguidamente, foi aplicado o método BIC para reduzir o número de componentes. Os resultados da aplicação deste critério aos vários modelos ACI são representados sob a forma apresentada na figura 6.1.

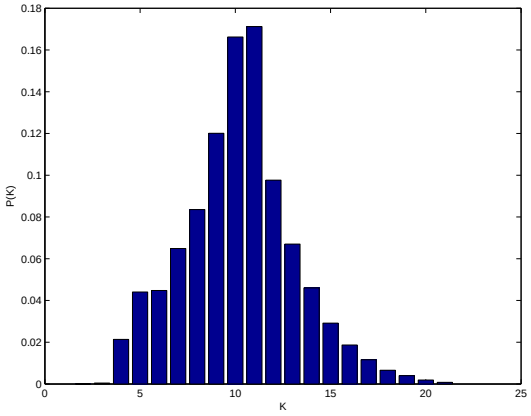


Figura 6.1: Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais.

Os valores de probabilidade presentes no "'array"' P apresentados na figura 6.1 são obtidos através da normalização dos valores de índice *bic* obtidos para cada modelo (número de componentes diferente). Esta normalização é feita da seguinte forma, considerando um "'array"' B contendo os valores de índice *bic*, o "'array"' P é obtido através das seguintes operações:

$$P(K) = e^{\frac{B(K)-max(B)}{N}}, \tag{6.1}$$

$$P(K) = \frac{P(K)}{\sum P}, \tag{6.2}$$

onde *max* é uma função que retorna o maior valor do "'array"', N o número de instantes e K o modelo (número de componentes).

Após aplicação do BIC, foi novamente aplicado o algoritmo de ACI mas separando apenas o número de componentes mais provável (anteriormente obtido pelo método BIC). Em seguida, foram de novo escolhidas as componentes independentes tendo em conta a sua latência.

Depois de seleccionadas as componentes, o passo seguinte foi o cálculo das topografias cerebrais e a localização das fontes neuronais para cada componente. Esta localização foi feita utilizando como modelo para a fonte o modelo de dipolo único para o $P3a$ e para o $P3b$, e o modelo de múltiplos dipolos (mais especificamente dois dipolos) para o potencial $N100$. Para o modelo da cabeça vão ser usados o modelo esférico e o modelo realista padrão.

O potencial $P300$ e, consequentemente, as suas componentes são o resultado de muitos geradores neuronais activos e localizados em áreas distintas do cérebro no instante em estudo. Deste modo, deve esclarecer-se que as técnicas de localização serviram apenas como métodos de parametrização, e não de localização. Assim, o modelo de dipolo único deve ser apenas interpretado como uma aproximação ao centro de massa formado pelos geradores neuronais activos. A utilização do modelo de múltiplos dipolos para o $N100$ deve-se ao facto deste potencial ter dois máximos negativos nas zonas temporais medial e frontal, cada um dos quais referente a um dos dois geradores neuronais que lhe estão subjacentes.

O sistema de eixos (espaço tridimensional) utilizado pelos modelo realista é representado na figura 6.2.

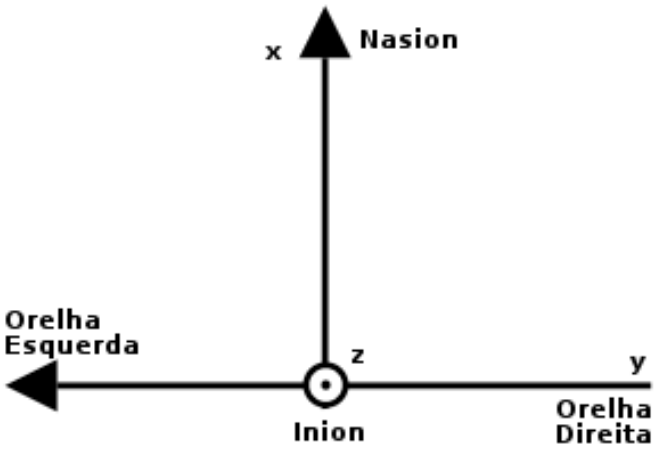


Figura 6.2: Sistema de eixos utilizado nos modelos esférico e realista.

A origem do sistema de eixos da figura 6.2 é o inion, com o sentido positivo do eixo dos xx do inion para o nasion, com o sentido positivo do eixo dos yy da direita para a esquerda e o sentido positivo do eixo dos zz de baixo para cima (profundidade)[1]. Nas figuras que irão ser apresentadas, as coordenadas das posições e orientações utilizam este sistema de eixos.

O modelo esférico considerado resulta de uma aproximação do crânio a uma esfera. Em [50], através de um método de convergência iterativo que minimiza a distância da superfície de uma esfera às posições dos eléctrodos, chegou-se à conclusão que a posição aproximada do centro da esfera que modela o crânio tem as coordenadas: $(8, 0, 3)$ (cm) e tem o raio de 7.8cm [50]. Estes foram os parâmetros utilizados no

[1] O modelo esférico utilizado neste trabalho utiliza um sistema de eixos diferente (considera a origem no centro do encéfalo), porém, todos os resultados obtidos com este modelo são transformados para o sistema de eixos com origem no inion.

modelo esférico utilizado neste trabalho. Com base nas espessuras médias dadas em [16] criaram-se, então, quatro esferas representando o encéfalo, o líquido céfalo-raquidiano, o crânio e o escalpe com raios de, respectivamente, 7.0cm, 7.2cm, 7.8cm e 8.2cm. As condutividades utilizadas para as diferentes estruturas foram: encéfalo - $0.33\Omega^{-1}m^{-1}$; líquido céfalo-raquidiano - $1\Omega^{-1}m^{-1}$; crânio - $0.0042\Omega^{-1}m^{-1}$ e escalpe - $0.33\Omega^{-1}m^{-1}$ [16].

O modelo realista considerado neste trabalho é constituído por três superfícies representando o encéfalo, crânio e escalpe. As condutividades utilizadas para as diferentes estruturas foram: encéfalo - $0.33\Omega^{-1}m^{-1}$; crânio - $0.01\Omega^{-1}m^{-1}$ e escalpe - $0.33\Omega^{-1}m^{-1}$ [50].

No processo de localização dipolar, o acordo entre os potenciais medidos e os potenciais criados pelo(s) dipolo(s) é quantificado através da variância residual (*RV*) para o caso do modelo esférico [9] (os valores abaixo dos 0.200 são empiricamente considerados como bons acordos) e o valor de delta (desvio Δ) para o modelo realista (os valores abaixo dos 0.400 são empiricamente considerados como bons acordos). O valor de delta é obtido através da expressão 6.3 [50].

$$\Delta = \frac{\sum_{i=1}^{N}(V_i - \Psi_i)^2}{\sum_{i=1}^{N}\Psi_i^2 - \frac{1}{N}(\sum_{i=1}^{N}\Psi_i)^2}, \tag{6.3}$$

onde V_i é o potencial calculado no eléctrodo i e Ψ_i é o potencial medido no eléctrodo i.

Para calcular as topografias cerebrais e a localização das fontes neuronais usando o modelo esférico, foi utilizado o *EEGLab* usando o *plugin Dipfit* desenvolvido por Robert Oostenveld [9], ambos disponíveis em [2]. Para a localização utilizando o modelo realista foi utilizado o programa *Dipoli* e para a sua visualização o programa *QTriplot* [8], ambos desenvolvidos por Thom Oostendorp [11]. As posições são dadas no formato (x, y, z), tendo como unidade o centímetro. As orientações são dadas no mesmo formato tendo como unidade o $\mu A.m$. As coordenadas das posições e das orientações dos dipolos estão colocadas nas figuras que vão ser apresentadas.

No modelo realista, utilizando o programa *Dipoli*, é necessário dar uma estimativa inicial da posição do dipolo. Esta posição inicial é muito importante para o cálculo do resultado final. Para ajudar a escolher a melhor estimativa inicial foi desenvolvido um programa em *Python*, que utiliza o *Dipoli*, e percorre um conjunto de posições iniciais definido pelo utilizador (por defeito é percorrido todo o volume descrito pela superfície que representa o encéfalo, de modo que cada posição está separada das restantes por, pelo menos, 1cm, numa das componentes x, y ou z). Um resultado típico deste programa é mostrado na figura 6.3.

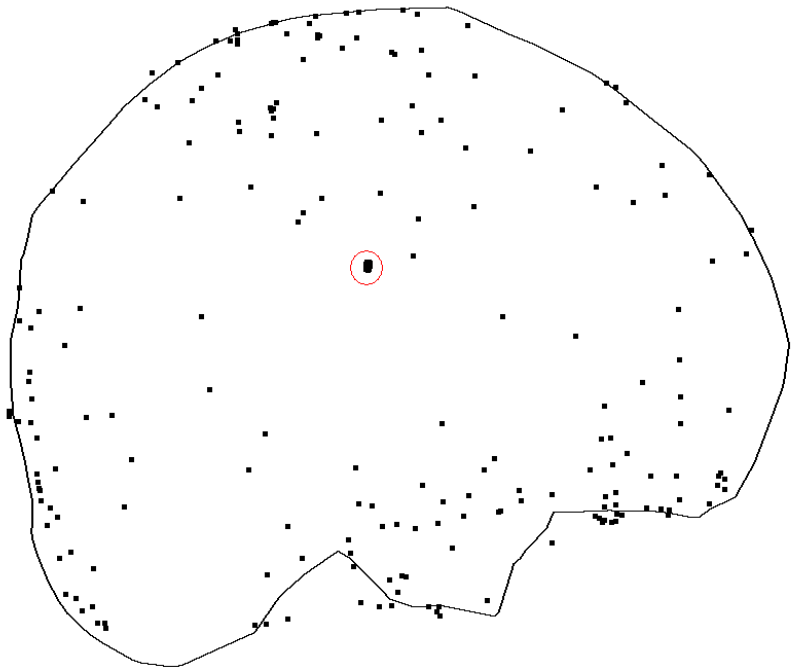


Figura 6.3: Localização dipolar utilizando várias estimativas iniciais para um potencial *P*300 de um indivíduo normal. O círculo vermelho na imagem indica o local onde existe um grande número de pontos, cerca de 900 neste caso, grande parte deles sobrepostos. Esta deverá ser a localização do gerador neuronal para este caso.

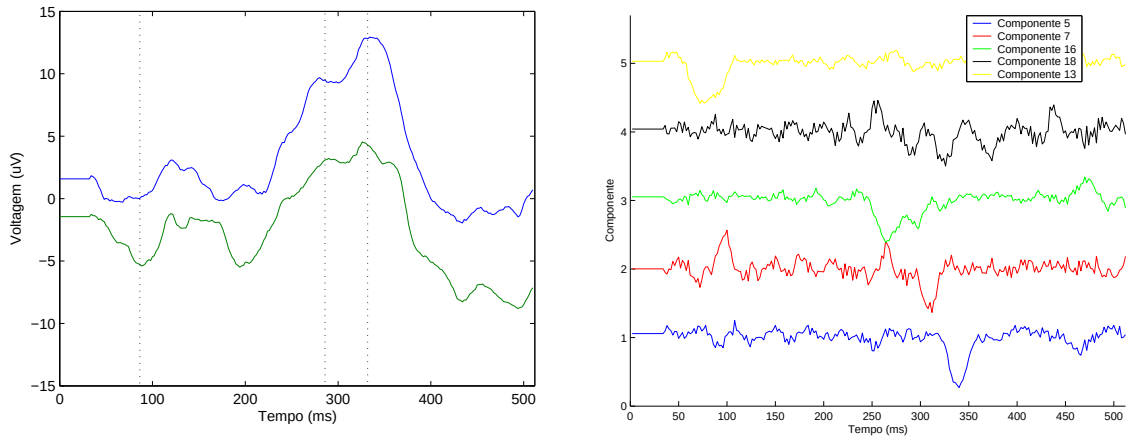
Como se pode ver na figura 6.3, uma má estimativa inicial pode conduzir a um resultado bastante erróneo. Uma grande convergência de pontos num determinado local indica que a melhor estimativa inicial se encontra naquela zona. A melhor estimativa inicial é, assim, aquela cujo resultado obteve o menor delta e que pertença ao grupo de maior convergência.

6.3 Indivíduos Normais

6.3.1 Normal 1

6.3.1.1 Identificação através das componentes calculadas pela ACI para a totalidade das componentes

Utilizando o método de envelope (explicado anteriormente) e aplicando o método de ACI anteriormente escolhido para o cálculo do número máximo de componentes (neste caso 21) e escolhendo aquelas que apresentam sinais com latências próximas de 100ms e de 300ms, obtém-se as figuras apresentadas em 6.4.



(a) (b)

Figura 6.4: a) Envelope dos máximos e mínimos dos potenciais. b) Componentes independentes escolhidas (Normal 1).

Analisando a figura 6.4(a), pode verificar-se pelas curvas de envelope a existência de duas componentes para o $P300$, uma localizada nos 281ms (curva dos mínimos) e a outra nos 337ms (curva dos máximos), picos perfeitamente distintos na mesma janela temporal, que poderão corresponder ao $P3a$ e $P3b$, respectivamente. Observa-se também um pico negativo localizado nos 90ms que poderá corresponder ao $N100$ (curva dos mínimos). Quanto à figura 6.4(b), foram escolhidas as componentes com picos nas janelas temporais anteriormente referidas: quatro componentes para o $P300$ (componentes 5, 7, 16 e 18) e uma componente para o $N100$ (componente 13). As topografias da actividade eléctrica e a localização das fontes foram calculadas e são apresentadas na figura 6.5.

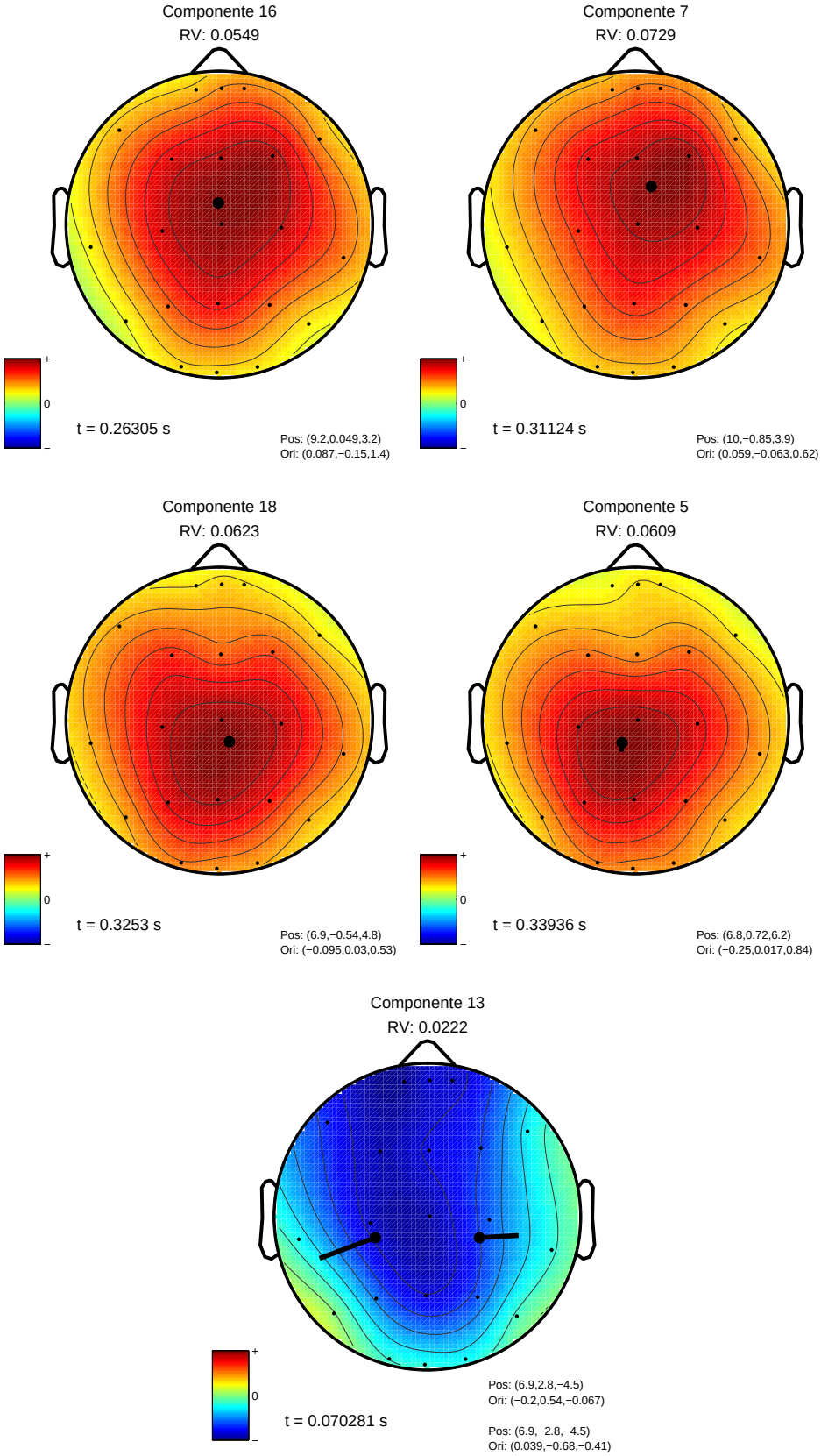


Figura 6.5: Mapa topográfico e localização dos dipolos referentes aos potenciais gerados pelas diferentes componentes, nos instantes de amplitude máxima na latência de interesse e usando o modelo esférico (Normal 1).

Componente 16	Componente 7	Componente 18	Componente 5
$P3a_1$	$P3a_2$	$P3b_1$	$P3b_2$
263ms	311ms	325ms	339ms

Tabela 6.1: Latências das componentes referentes ao $P300$ (Normal 1).

Tendo em conta que a sub-componente $P3a$ deve preceder a $P3b$ e que a primeira apresenta uma localização central mais frontal que a segunda é possível, através da análise da figura 6.5 e da tabela 6.1, verificar que:

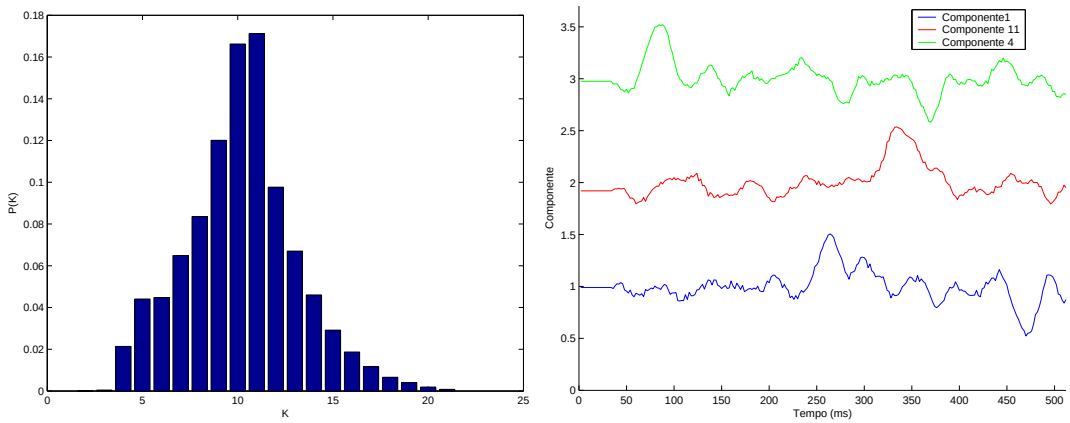
- A componente 7 poderá corresponder ao potencial $P3a$ devido à latência do pico próxima dos 300ms, devido à actividade eléctrica mais concentrada na zona frontal (Fz) e devido à localização do dipolo que se encontra situado entre as zonas central e frontal. A componente 16 poderá corresponder ao potencial $P3a$ devido à latência do pico, devido à actividade eléctrica mais concentrada na zona frontal (Fz) e a localização do dipolo que está situado entre as zonas central e frontal, deverá representar o $P3a$.
- As componentes 18 e 5 poderão corresponder ao potencial $P3b$ devido à latência do pico, devido à actividade eléctrica mais concentrada na zona parietal (Pz) e devido à localização do dipolo que se encontra entre as zonas central e parietal.

Desta forma, tanto o $P3a$ como o $P3b$ foram identificados por duas sub-componentes cada um, ocorrendo, portanto, sub-divisão das componentes de interesse.

Com latência de 70ms, a componente 13 poderá representar o $N100$. Esta apresenta uma actividade eléctrica concentrada na zona frontal (Fz). Os dipolos correspondentes estão situados na zona temporo-parietal inequivocamente lateralizados para a esquerda e direita e com orientações para às zonas temporal esquerda e direita respectivamente.

6.3.1.2 Identificação através das componentes calculadas pela ACI após redução das componentes

Aplicando o método BIC para estimar o número de fontes presentes nos registos do indivíduo normal 1 obtém-se as probabilidades ilustradas na figura 6.6(a). Aplicando de novo o algoritmo de ACI mas procurando apenas 11 componentes, conforme sugerido pelo método anterior, as componentes escolhidas foram as apresentadas na figura 6.6(b).



(a) (b)

Figura 6.6: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes escolhidas após redução do seu número para 11 (Normal 1).

Como se pode verificar, o ruído foi consideravelmente reduzido e em vez de cinco componentes, passa-se a ter três componentes de interesse para este estudo, duas para o $P300$ e uma para o $N100$. Além disso, é curioso verificar que os máximos (mínimos) observados nas componentes assemelham-se mais aos correspondentes máximos (mínimos) observados visualmente nos registos originais. Os mapas topográficos destas componentes e localizações das fontes neuronais para o modelo esférico e realista padrão são mostradas na figura 6.7.

Componente 1	**Componente 11**	**Componente 4**
$P3a$	$P3b$	$N100$
$263ms$	$333ms$	$86ms$

Tabela 6.2: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (Normal 1).

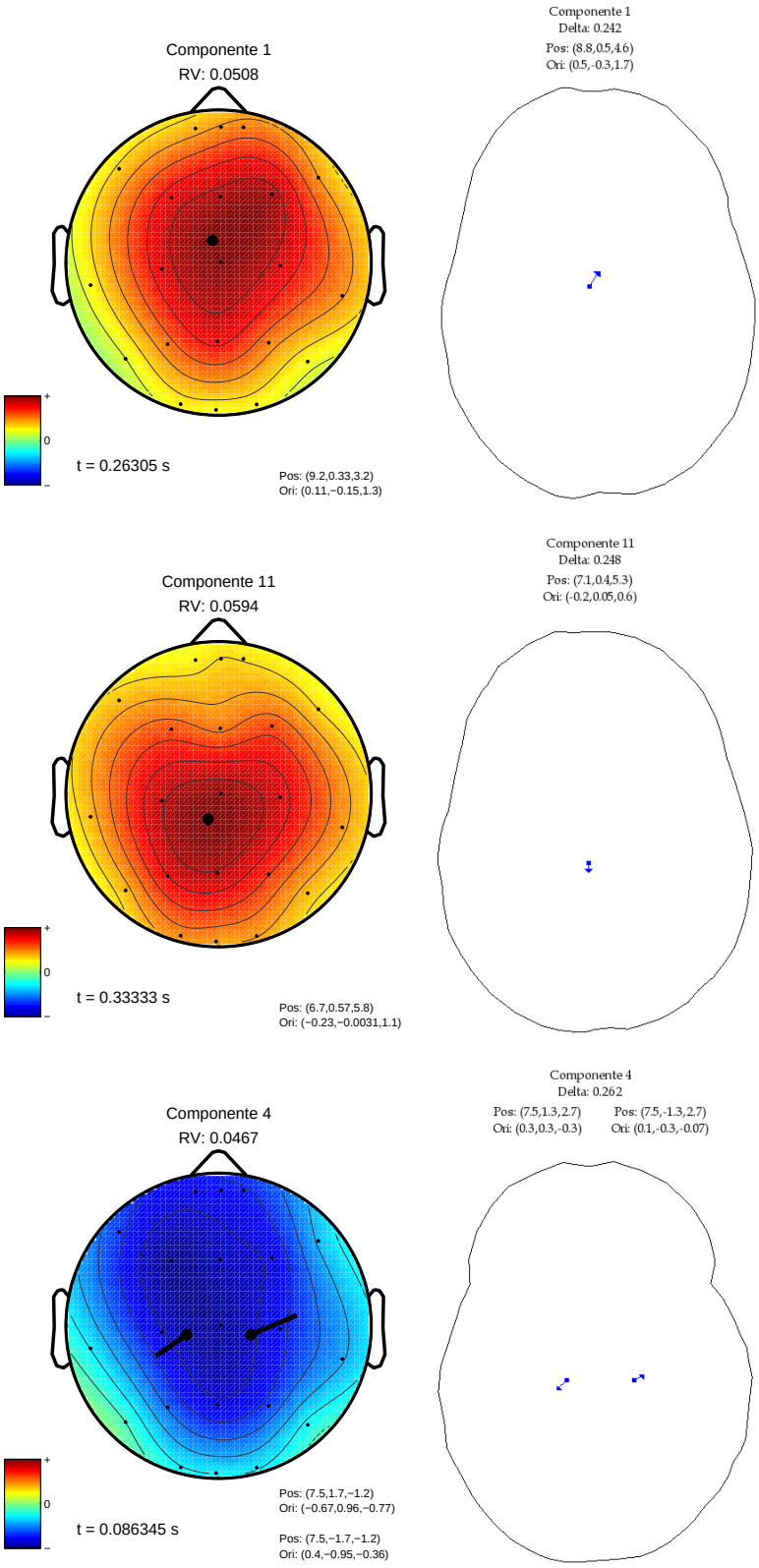


Figura 6.7: Mapa topográfico e localização das componentes escolhidas após redução do número de componentes procuradas (lado esquerdo - modelo esférico, lado direito - modelo realista) (Normal 1).

Componente 1	
Latência: 263**ms**	
ID: *P3a*	
Modelo Esférico	
RV: 0.0508	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.2,0.33,3.2)	(0.11,-0.15,1.3)
Modelo Realista	
Delta (Δ): 0.242	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.8,0.5,4.6)	(0.5,-0.3,1.7)

Componente 11	
Latência: 333**ms**	
ID: *P3b*	
Modelo Esférico	
RV: 0.0594	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.7,0.57,5.8)	(-0.23,-0.0031,1.1)
Modelo Realista	
Delta (Δ): 0.248	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.1,0.4,5.3)	(-0.2,0.05,0.6)

Componente 4	
Latência: 86**ms**	
ID: *N100*	
Modelo Esférico	
RV: 0.0467	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.5,1.7,-1.2)	(-0.67,0.96,-0.77)
(7.5,-1.7,-1.2)	(0.4,-0.95,-0.36)
Modelo Realista	
Delta (Δ): 0.262	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.5,1.3,2.7)	(0.3,0.3,-0.3)
(7.5,-1.3,2.7)	(0.1,-0.3,-0.07)

Tabela 6.3: Características das componentes escolhidas (Normal 1).

Analisando a figura 6.7 e a tabela 6.3 pode-se verificar que:

- A componente 1 poderá representar o $P3a$ devido à latência do pico positivo, devido à actividade eléctrica concentrada na zona frontal (Fz), devido à localização do dipolo que está situado entre as zonas central e frontal e devido à sua orientação, obtida no modelo realista. O dipolo está orientado para a zona frontal.

- A componente 11 poderá representar o $P3b$ devido à latência do pico positivo, devido à actividade eléctrica concentrada na zona parietal (Pz), devido à localização do dipolo que se encontra situado entre as zonas central e parietal e devido à sua orientação, obtida no modelo realista. O dipolo está orientado para a zona parietal.

Localizada e apresentada na figura 6.7 e com latência de 86ms, a componente 4 poderá representar o $N100$. Esta apresenta uma actividade eléctrica concentrada na zona frontal (Fz). Os dipolos estão situados na zona central lateralizados para a esquerda e direita e com orientações com direcção oblíqua e sentido para as zonas temporal esquerda e direita, respectivamente.

6.3.2 Normal 2

Para o segundo indivíduo pertencente ao grupo normal, procedeu-se de forma idêntica à anteriormente explanada e foram obtidos os resultados que em seguida se apresenta.

6.3.2.1 Identificação através da latência nos sinais originais

Aplicando o método de envelope dos potenciais a este caso, as latências identificadas onde provavelmente se encontram os potenciais $P3a$, $P3b$ e $N100$ estão presentes na tabela 6.4.

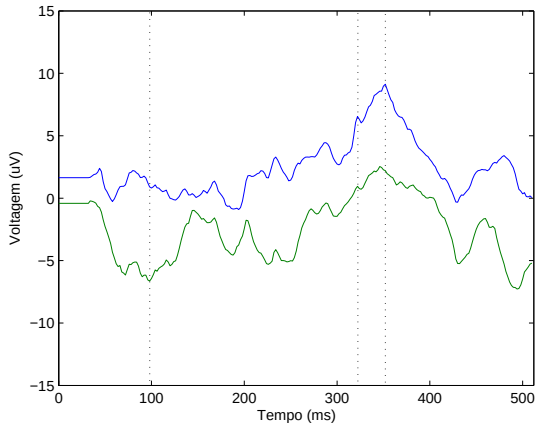


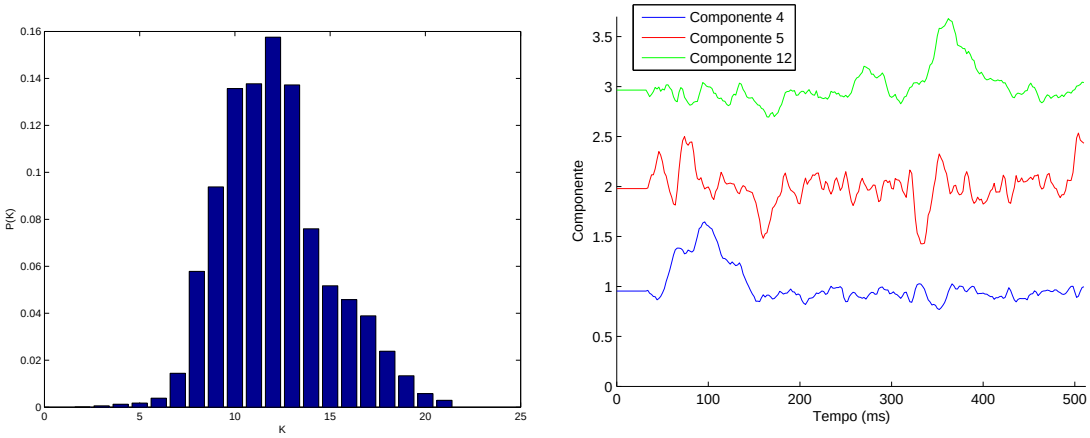
Figura 6.8: Envelope dos potenciais (Normal 2).

Potenciais	$P3a$	$P3b$	$N100$
Latências	323ms	353ms	99ms
Envelope	Máximos	Máximos	Mínimos

Tabela 6.4: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (classificação através da latência) (Normal 2).

6.3.2.2 Identificação através das componentes calculadas pela ACI

Utilizando o critério BIC, verificou-se que para este caso, o número mais provável de fontes presentes nos dados originais é de 12 (ver figura 6.9(a)). Aplicando o algoritmo de ACI procurando este número de componentes, foram escolhidas as componentes apresentadas na figura 6.9(b) como possíveis representantes dos potenciais $P3a$, $P3b$ e $N100$.



(a) (b)

Figura 6.9: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 12 componentes (Normal 2).

Componente 5	**Componente 12**	**Componente 4**
$P3a$	$P3b$	$N100$
337ms	363ms	96ms

Tabela 6.5: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (Normal 2).

Os mapas topográficos destas componentes e suas localizações usando o modelo esférico e modelo realista padrão são mostrados na figura 6.10.

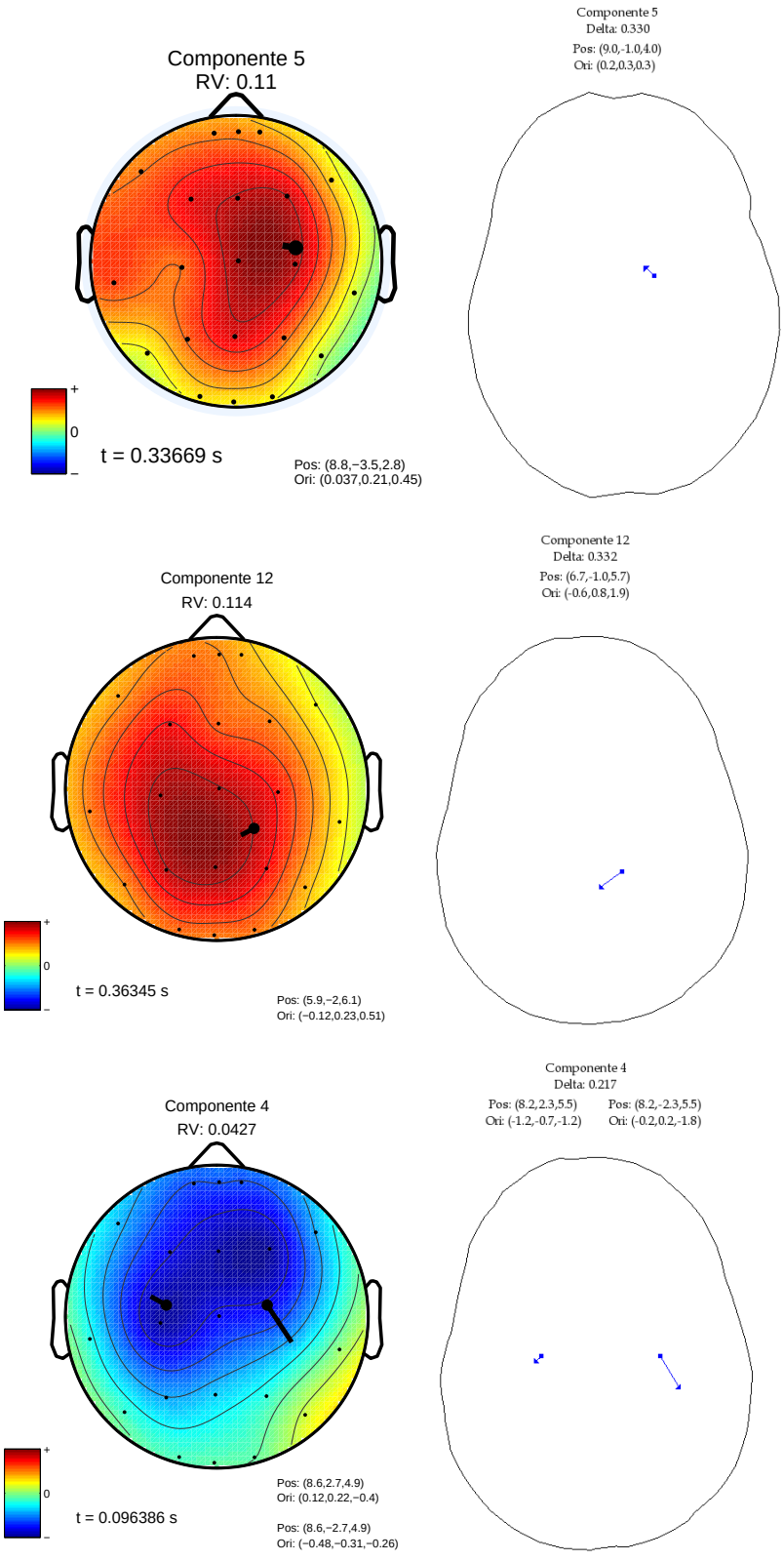


Figura 6.10: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Normal 2).

Componente 5	
Latência: 337**ms**	
ID: *P3a*	
Modelo Esférico	
RV: 0.11	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.8,-3.5,2.8)	(0.037,0.21,0.45)
Modelo Realista	
Delta (Δ): 0.330	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.0,-1.0,4.0)	(0.2,0.3,0.3)

Componente 12	
Latência: 363**ms**	
ID: *P3b*	
Modelo Esférico	
RV: 0.114	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(5.9,-2.0,6.1)	(-0.12,0.23,0.51)
Modelo Realista	
Delta (Δ): 0.332	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.7,-1.0,5.7)	(-0.6,0.8,1.9)

Componente 4	
Latência: 96**ms**	
ID: *N100*	
Modelo Esférico	
RV: 0.0427	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.6,2.7,4.9)	(0.12,0.22,-0.4)
(8.6,-2.7,4.9)	(-0.48,-0.31,-0.26)
Modelo Realista	
Delta (Δ): 0.217	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.2,2.3,5.5)	(-1.2,-0.7,-1.2)
(8.2,-2.3,5.5)	(-0.2,0.2,-1.8)

Tabela 6.6: Características das componentes escolhidas (Normal 2).

Analisando a figura 6.10 e a tabela 6.6 pode-se verificar que:

- A componente 5 poderá representar o $P3a$ devido à latência do pico e devido à actividade eléctrica concentrada na zona frontal (Fz). Quanto à localização do dipolo, este está situado entre as zonas central e frontal com orientação para a zona frontal em ambos os modelos. Esta posição e orientação ajudam a identificar esta componente como sendo o $P3a$. O dipolo encontra-se ligeiramente deslocado para a direita, principalmente no modelo esférico.

- A componente 12 poderá representar o $P3b$ devido à latência do pico, devido à actividade eléctrica concentrada na zona parietal (Pz) e devido à localização do dipolo que está situado entre as zonas central e parietal, ligeiramente deslocado para a direita e orientado para a zona parietal.

Pela figura 6.10, o pico da componente 4 tem uma latência de 96ms e assim poderá representar o $N100$. Esta componente tem uma actividade eléctrica concentrada na zona frontal (Fz) e central (Cz). Os dipolos estão situados na zona central lateralizados para a esquerda e direita, orientados para às zonas temporal esquerda e direita, respectivamente, como seria de esperar.

Analisando com atenção as componentes 5 e 12, verificou-se que as topografias da actividade eléctrica destas componentes apresentam assimetrias significativas e os dipolos calculados apresentam localizações ligeiramente lateralizadas para a direita com orientações lateralizadas, principalmente quando é utilizado o modelo esférico.

Tratando-se este caso de um indivíduo normal, este tipo de lateralização dos dipolos não deveria acontecer, fugindo assim este caso da "média". De referir que esta situação verificou-se também nas componentes obtidas através do cálculo do número máximo de componentes como também no estudo [23] que analisa o mesmo caso, sendo uma observação para aqual não se encontrou justificação a não ser, quanto muito, a possibilidade de uma má colocação dos eléctrodos.

6.3.3 Normal 3

Os resultados que se seguem são referentes ao 3º indivíduo normal e foram obtidos através do procedimento anteriormente descrito.

6.3.3.1 Identificação através da latência nos sinais originais

Analisando o resultado da aplicação do método de envelope dos potenciais, as latências onde provavelmente se encontram os potenciais em estudo estão presentes na tabela 6.7.

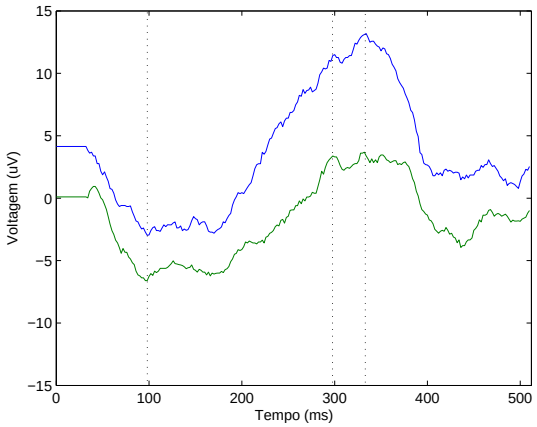


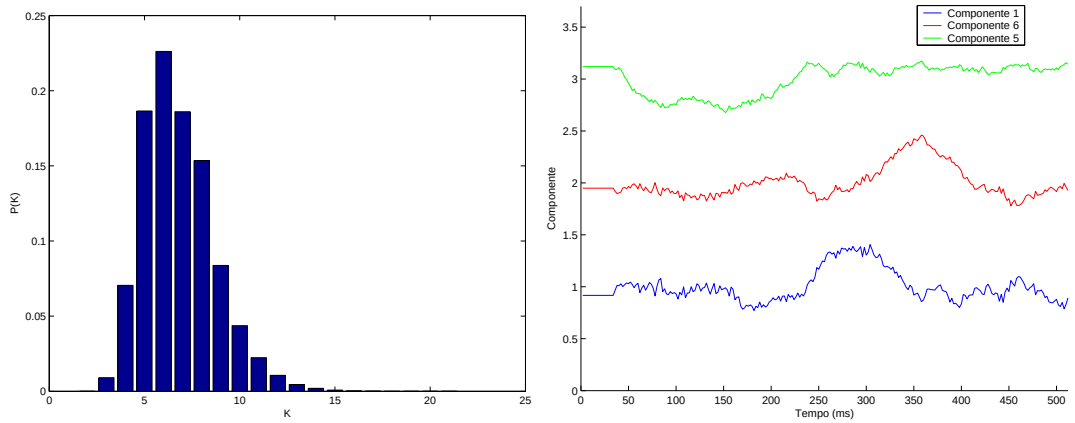
Figura 6.11: Envelope dos potenciais (Normal 3).

Potenciais	*P*3*a*	*P*3*b*	*N*100
Latências	290ms	335ms	98ms
Envelope	Máximos	Máximos	Mínimos

Tabela 6.7: Latências de cada um dos potenciais (Normal 3).

6.3.3.2 Identificação através das componentes calculadas pela ACI

Após a aplicação do critério BIC, como se pode ver na figura 6.12(a), o número mais provável de fontes presentes nos dados é seis. Utilizando o algoritmo de ACI na procura deste número de componentes, foram escolhidas as componentes apresentadas na figura 6.12(b) como possíveis representantes dos potenciais *P*3*a*, *P*3*b* e *N*100.



(a) (b)

Figura 6.12: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 6 componentes (Normal 3).

Componente 1	**Componente 6**	**Componente 5**
$P3a$	$P3b$	$N100$
303ms	357ms	88ms

Tabela 6.8: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (classificação através da latência) (Normal 3).

Os mapas topográficos destas componentes e suas localizações usando o modelo esférico e modelo realista padrão são mostrados na figura 6.13.

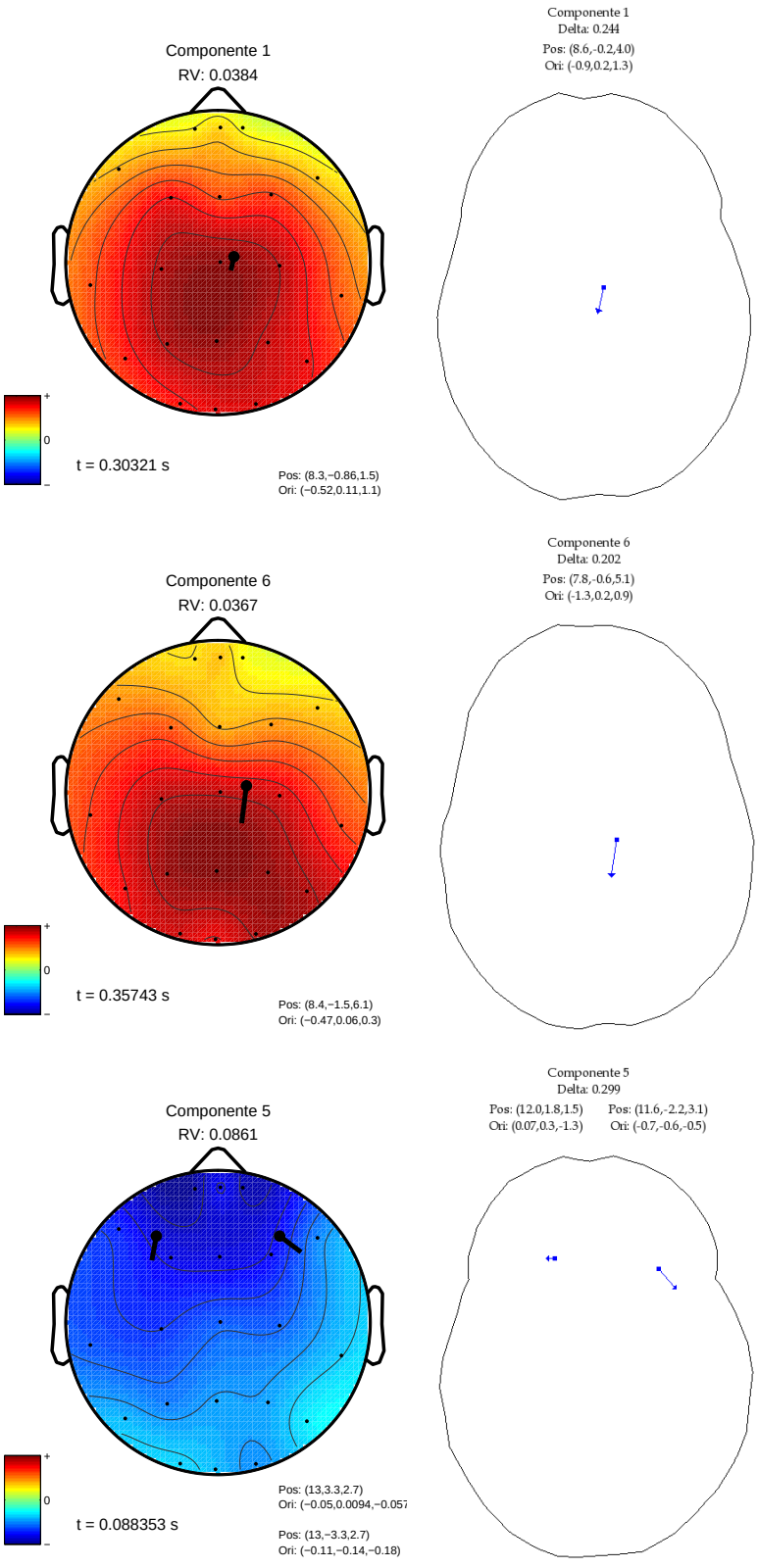


Figura 6.13: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Normal 3).

Componente 1	
Latência: 303**ms**	
ID: *P3a*	
Modelo Esférico	
RV: 0.0384	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.3,-0.86,1.5)	(0.52,0.11,1.1)
Modelo Realista	
Delta (Δ): 0.244	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.6,-0.2,4.0)	(-0.9,0.2,1.3)

Componente 6	
Latência: 357**ms**	
ID: *P3b*	
Modelo Esférico	
RV: 0.0367	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.4,-1.5,6.1)	(-0.47,0.06,0.3)
Modelo Realista	
Delta (Δ): 0.202	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.8,-0.6,5.1)	(-1.3,0.2,0.9)

Componente 5	
Latência: 88**ms**	
ID: *N100*	
Modelo Esférico	
RV: 0.0861	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(13,3.3,2.7)	(-0.05,0.01,-0.06)
(13,-3.3,2.7)	(-0.11,-0.14,-0.18)
Modelo Realista	
Delta (Δ): 0.299	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(12.0,1.8,1.5)	(0.07,0.3,-1.3)
(11.6,-2.2,3.1)	(-0.7,-0.6,-0.5)

Tabela 6.9: Características das componentes escolhidas (Normal 3).

Analisando a figura 6.13 e a tabela 6.9 pode-se verificar que:

- A componente 1 poderá representar o $P3a$ devido à latência do pico. No modelo realista, verifica-se que o dipolo está situado numa posição ligeiramente anterior (entre a zona central e frontal) em relação ao dipolo da componente 6 o que reforça a assunção de que esta componente pode representar o $P3a$. A orientação do dipolo e a actividade eléctrica concentrada entre a zona central (Cz) e a zona parietal (Pz) poderá, no entanto, denotar já a presença do potencial $P3b$. Este aspecto é, aliás, corroborado pela análise da figura 6.11 onde não é possível verificar uma clara separação entre o potencial $P3a$ e $P3b$.

- A componente 6 poderá representar o $P3b$ devido à latência do pico, devido à actividade eléctrica concentrada na zona parietal (Pz) e devido à localização do dipolo que está situada na zona central, ligeiramente deslocado para a direita, orientado para a zona parietal.

Analisando a figura 6.13 e com latência de 88ms, a componente 5 poderá representar o $N100$. Esta componente tem uma actividade eléctrica concentrada na zona frontal (Fz) e central (Cz). Os dipolos estão situados na zona frontal lateralizados para a esquerda e direita, orientados para às zonas temporal esquerda e direita respectivamente, encontrando-se, no entanto, em posições excessivamente frontais relativamente ao que seria de esperar. Como já foi referido, o potencial $N100$ está relacionado com a simples audição do estímulo, pelo que a sua origem deveria encontrar-se na área auditiva primária, que se localiza na região temporal. Supõe-se que esta dificuldade em localizar as fontes na região mais provável se possa dever ao baixo número de eléctrodos. Seria de esperar que a existência de apenas 21 eléctrodos dificultasse a procura de mais do que uma fonte, não é de estranhar que em alguns casos essa dificuldade se torne mais presente.

6.3.4 Normal 4

Os resultados referentes ao indivíduo 4 do grupo normal foram processados de forma similar à dos anteriores e encontram-se descritos em seguida.

6.3.4.1 Identificação através da latência nos sinais originais

Após a aplicação do método de envelope dos potenciais, as latências identificadas onde provavelmente se encontram os potenciais $P3a$, $P3b$ e $N100$ estão presentes na tabela 6.10.

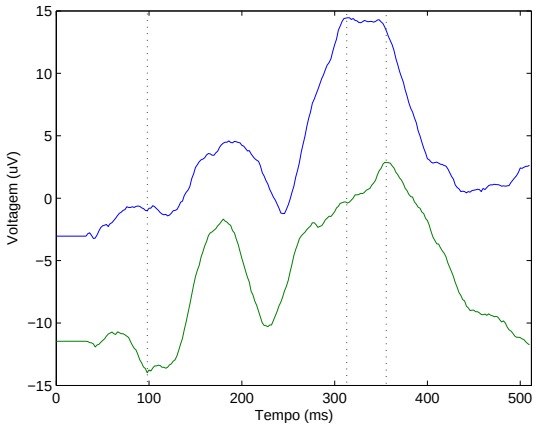


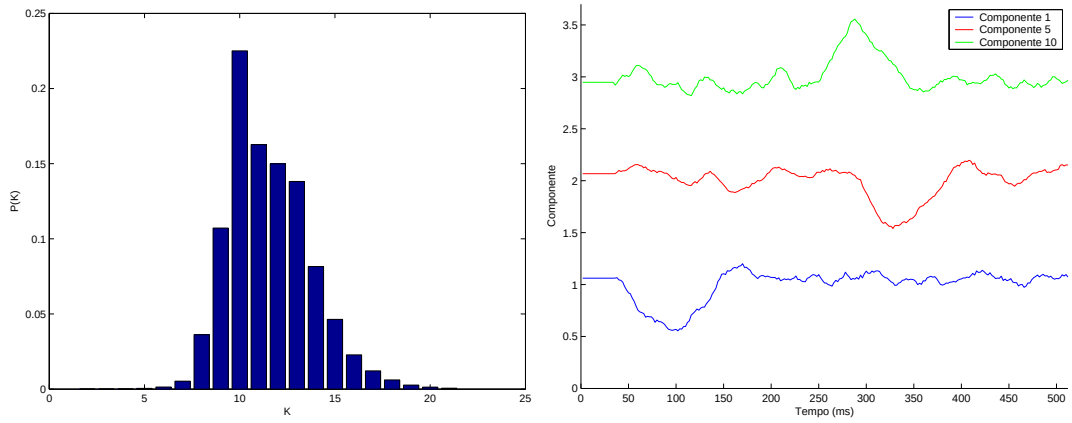
Figura 6.14: Envelope dos potenciais (Normal 4).

Potenciais	$P3a$	$P3b$	$N100$
Latências	312ms	355ms	98ms
Envelope	Máximos	Mínimos	Mínimos

Tabela 6.10: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (classificação através da latência) (Normal 4).

6.3.4.2 Identificação através das componentes calculadas pela ACI

Utilizando o método BIC, o número mais provável de fontes presentes nos dados originais é de 10 (ver figura 6.15(a)). Aplicando o algoritmo de ACI procurando 10 componentes, foram escolhidas as componentes apresentadas na figura 6.15(b) como possíveis representantes dos potenciais $P3a$, $P3b$ e $N100$.



(a) (b)

Figura 6.15: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 10 componentes (Normal 4).

Componente 10	**Componente 5**	**Componente 1**
$P3a$	$P3b$	$N100$
287ms	327ms	100ms

Tabela 6.11: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (Normal 4).

Os mapas topográficos destas componentes e suas localizações usando o modelo esférico e modelo realista padrão são mostrados na figura 6.16.

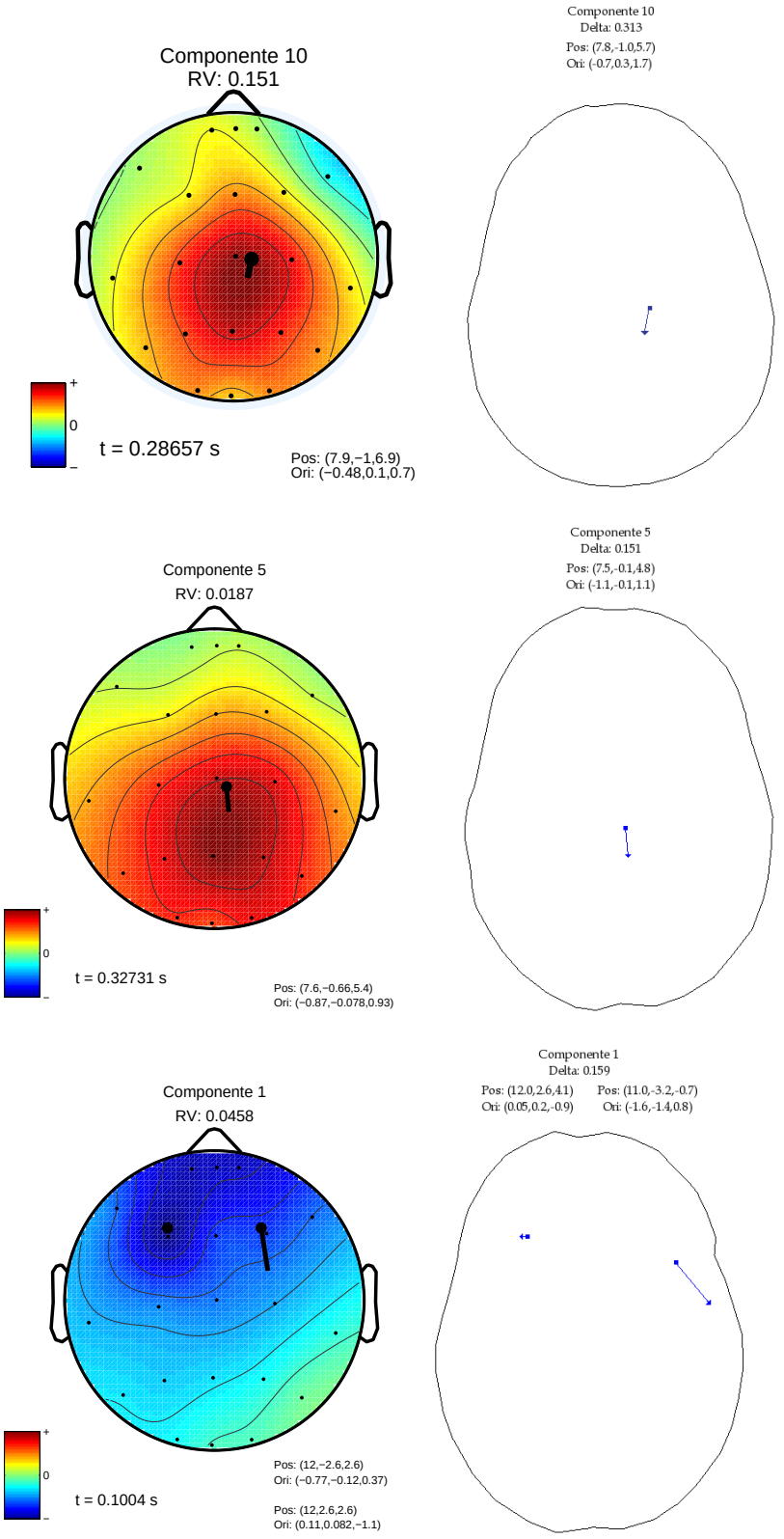


Figura 6.16: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Normal 4).

Componente 10	
Latência: 287**ms**	
ID: *P3a*	
Modelo Esférico	
RV: 0.151	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.9,-1.6,6.9)	(-0.48,0.1,0.7)
Modelo Realista	
Delta (Δ): 0.313	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.8,-1.0,5.7)	(-0.7,0.3,1.7)

Componente 5	
Latência: 327**ms**	
ID: *P3b*	
Modelo Esférico	
RV: 0.0187	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.6,-0.66,5.4)	(-0.87,-0.09,0.93)
Modelo Realista	
Delta (Δ): 0.151	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.5,-0.1,4.8)	(-1.1,-0.1,1.1)

Componente 1	
Latência: 100**ms**	
ID: *N100*	
Modelo Esférico	
RV: 0.0458	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(12,-2.6,2.6)	(-0.77,-0.12,0.37)
(12,2.6,2.6)	(0.11,0.08,-1.1)
Modelo Realista	
Delta (Δ): 0.159	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(12,2.6,4.1)	(0.05,0.2,-0.9)
(11,-3.2,-0.7)	(-1.6,-1.4,0.8)

Tabela 6.12: Características das componentes escolhidas (Normal 4).

Analisando a figura 6.16 e a tabela 6.12 pode-se verificar que:

- A componente 10 poderá representar o $P3a$ devido à latência do pico. No modelo realista, verifica-se que o dipolo está situado numa posição ligeiramente anterior (entre a zona central e frontal) em relação ao dipolo da componente 5 o que reforça a assunção de que esta componente pode representar o $P3a$. Esta situação é bastante similar à do Normal 3. A orientação do dipolo e a actividade eléctrica concentrada entre a zona central (Cz) e a zona parietal (Pz) poderá também denotar já a presença do potencial $P3b$. A partir da figura 6.14 e da tabela 6.10 seria já possível prever-se esta situação pelo facto de não haver uma clara separação entre o potencial $P3a$ e $P3b$.

- A componente 5 poderá representar o $P3b$ devido à latência do pico, devido à actividade eléctrica concentrada na zona parietal (Pz) e devido à localização do dipolo que está situado na zona central, com orientação para a zona parietal.

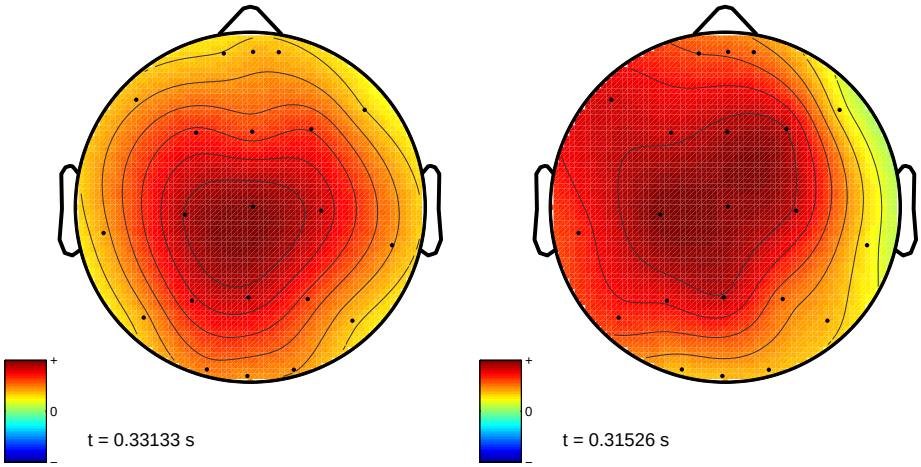
Pela figura 6.16, o pico da componente 1 tem uma latência de 100ms e, desta forma, poderá representar o $N100$. Esta componente tem uma actividade eléctrica concentrada na zona frontal (Fz) e central (Cz). Os dipolos, tal como no caso anterior, estão situados numa zona demasiado frontal para o que seria de esperar, porém, lateralizados para a esquerda e direita, denotando simetria. O dipolo direito está orientado para a zona central direita e o dipolo esquerdo orientado para a zona frontal esquerda.

6.4 Indivíduos Doentes

6.4.1 Doente 1

6.4.1.1 Introdução

Como foi demonstrado, a distribuição eléctrica do P300 ao nível do escalpe em indivíduos normais é normalmente central e simétrica. Porém, no caso de indivíduos doentes (epilepsias temporais) esta distribuição passa a ser, na maior parte dos casos, lateralizada. Esta situação encontra-se ilustrada nas figuras 6.17(a) e 6.17(b)



(a) Normal (b) Doente

Figura 6.17: Mapas topográficos dos potenciais originais *P*300 de um indivíduo normal e de um indivíduo doente.

Em trabalhos anteriores [23], verificou-se que, através da localização de fontes, para o *P*300 e suas componentes, poder-ia identificar se a epilepsia temporal era direita ou esquerda.

De notar que, devido a estas assimetrias dos mapas topográficos, tentou-se utilizar o modelo de dois dipolos na localização da fonte para as componentes do *P*300. Verificou-se, no entanto, que, para a maioria dos casos, os dipolos se apresentavam sobrepostos ou em posições muito próximas, obtendo-se, assim, uma caracterização mais eficaz através do uso do modelo de dipolo único. Desta forma, apenas casos onde se observou um mau acordo para o modelo de dipolo único, se apresentam os resultados utilizando o modelo de duplo dipolo.

6.4.1.2 Identificação do lado da epilepsia

Ao se processar a componente *P*300 original sem o recurso à análise em componentes independentes, obteve-se relativamente à posição de um único dipolo a seguinte distribuição para o caso do doente 1 que tem uma epilepsia temporal direita:

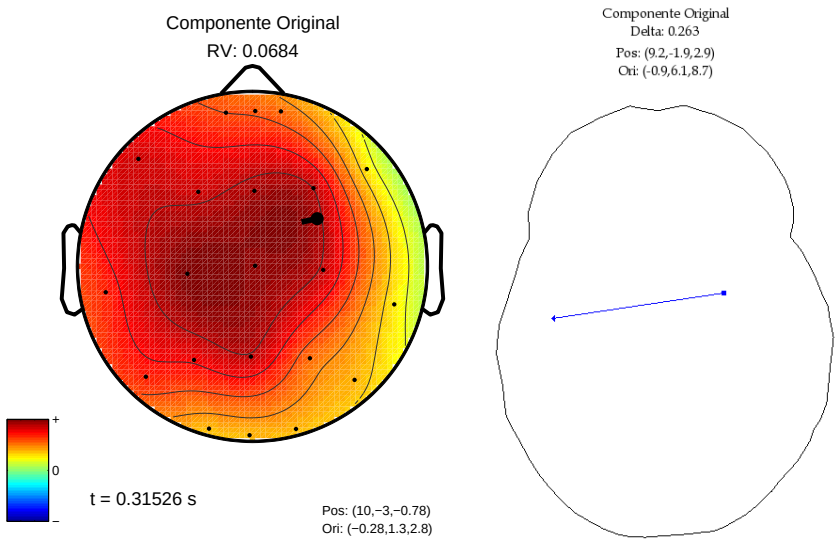


Figura 6.18: Mapa topográfico da componente P300 original e localização do dipolo (modelo esférico e realista). (Doente 1)

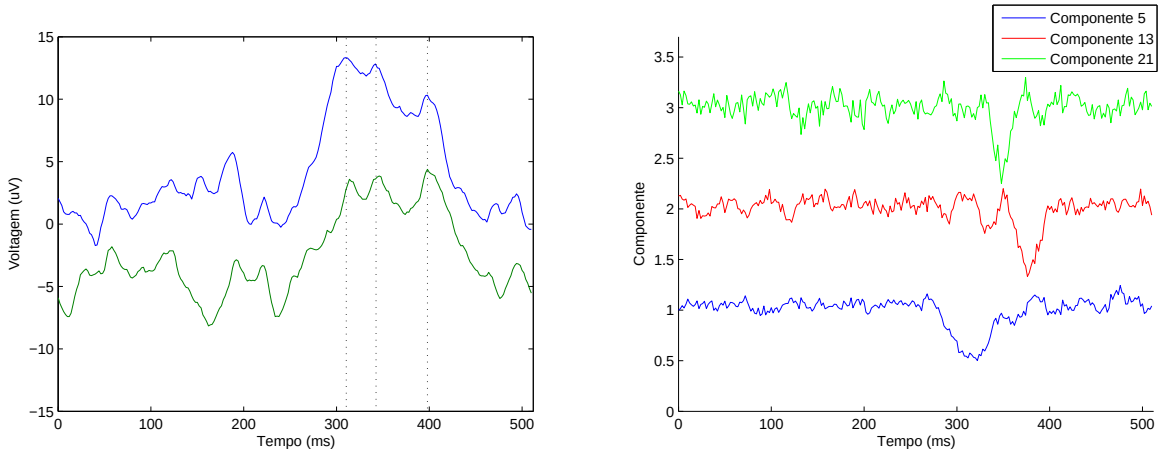
Componente Original	
Latência: 315**ms**	
ID: *P*300	
Modelo Esférico	
RV: 0.0684	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(10,-3,-0.78)	(-0.28,1.3,2.8)
Modelo Realista	
Delta (Δ): 0.263	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.2,-1.9,2.9)	(-0.9,6.1,8.7)

Tabela 6.13: Características da componente original (Doente 1).

Observando a localização do dipolo e pela forte orientação para a zona temporal esquerda (no caso do modelo realista), e sabendo-se que o indivíduo tem uma epilepsia temporal direita, pode conjecturar-se que focos de epilepsia lateralizados poderão induzir uma lateralização da posição do dipolo e da sua orientação. Denota-se neste caso, uma forte assimetria tanto na localização como na orientação para o lado contrário à localização da epilepsia.

6.4.1.3 Identificação através das componentes calculadas pela ACI para a totalidade das componentes

Em seguida vai ser demonstrada a presença de *overlearning* nas componentes do *P*300 mesmo em indíviduos doentes. Aplicando o método de envelope e utilizando o método de ACI para o cálculo do número máximo de componentes (21) e escolhendo aquelas que apresentam sinais com latências próximas de 300ms, obtém-se as figuras apresentadas em 6.19.



(a) (b)

Figura 6.19: a) Envelope dos máximos e mínimos dos potenciais. b) Componentes independentes escolhidas (Doente 1).

Analisando a figura 6.19(a), pode verificar-se pelas curvas de envelope, picos perfeitamente distintos localizados nas janelas temporais das componentes do *P*300, nos 316ms (curva dos máximos), nos 348ms (curva dos mínimos) e outro nos 390ms (curva dos máximos). Relativamente ao *N*100, não é possível identificar com toda a certeza nas curvas de envelope a latência deste potencial. Quanto à figura 6.19(b), foram escolhidas as componentes com picos nas janelas temporais anteriormente referidas: três componentes para o *P*300. As topografias da actividade eléctrica e a localização das fontes foram calculadas e são apresentadas na figura 6.20.

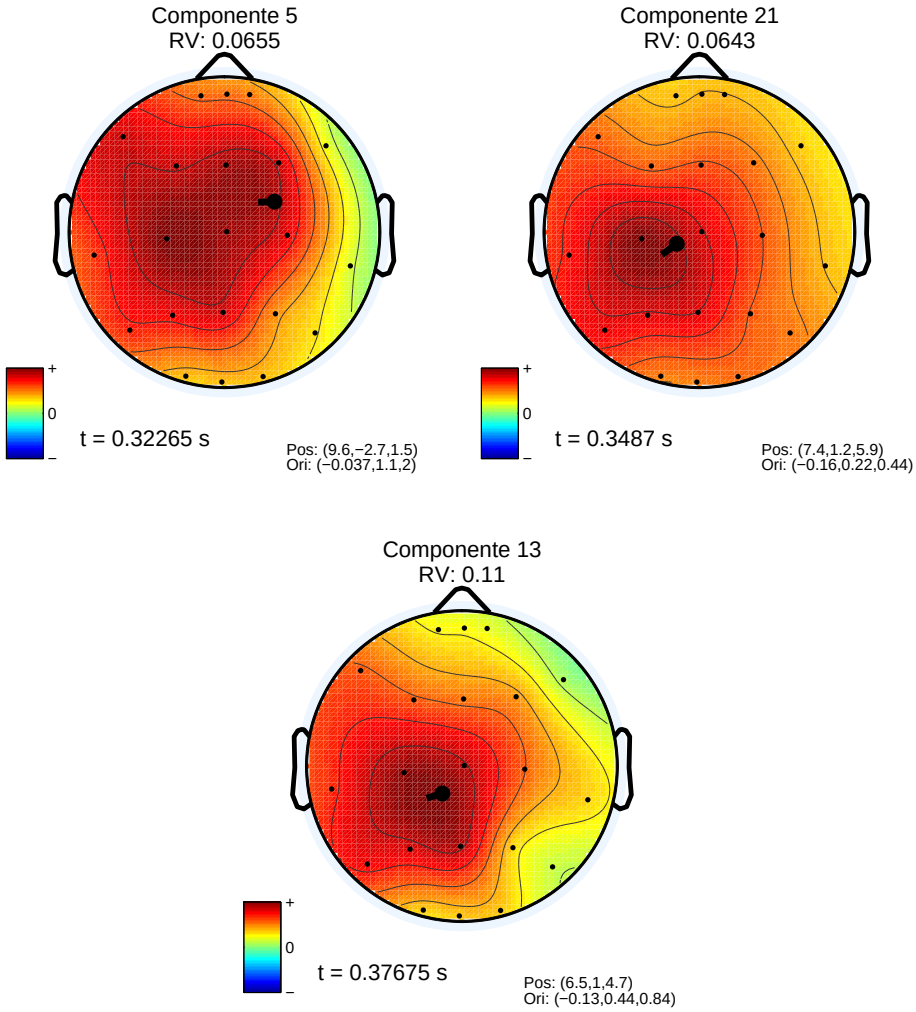


Figura 6.20: Mapa topográfico e localização dos dipolos referentes aos potenciais gerados pelas diferentes componentes, nos instantes de amplitude máxima na latência de interesse e usando o modelo esférico (Doente 1).

Componente 5	**Componente 21**	**Componente 13**
$P3a$	$P3b_1$	$P3b_2$
323ms	349ms	376ms

Tabela 6.14: Latências das componentes referentes ao $P3a$ e $P3b$ (Doente 1).

Tendo em conta que a sub-componente $P3a$ deve preceder a $P3b$ e que a primeira apresenta uma localização central mais frontal que a segunda é possível, através da análise da figura 6.20 e da tabela 6.14, verificar que:

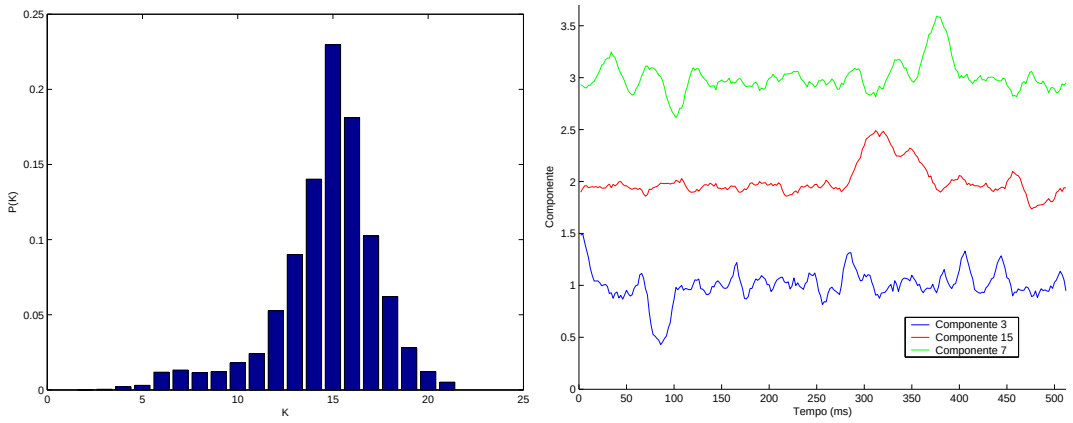
- A componente 5 poderá corresponder ao potencial $P3a$ devido à latência do pico próxima dos 300ms, devido à actividade eléctrica mais concentrada na zona frontal (Fz) e devido à localização do dipolo que se encontra situado entre as zonas central e frontal, lateralizado para a direita.

- As componentes 21 e 13 poderão corresponder ao potencial *P*3*b* devido à latência do pico, devido à actividade eléctrica mais concentrada na zona parietal (*Pz*) e devido à localização do dipolo que se encontra na zona central.

Devido à semelhante distribuição eléctrica e dipolos com posição e orientação também semelhantes, pode-se considerar a presença de *overlearning* no potencial *P*3*b*, nas componentes 21 e 13.

6.4.1.4 Identificação através das componentes calculadas pela ACI após redução das componentes

Aplicando o método BIC para estimar o número de fontes presentes nas misturas obtém-se as probabilidades ilustradas na figura 6.21(a). Aplicando de novo o algoritmo de ACI mas procurando apenas 15 componentes, as componentes escolhidas foram as apresentadas na figura 6.21(b).



(a) (b)

Figura 6.21: a) Probabilidade de adequação de cada modelo de ACI *K* (que denota o número de componentes) aos dados originais. b) Componentes escolhidas após redução do seu número para 15 (Doente 1).

Componente 15	Componente 7	Componente 3
*P*3*a*	*P*3*b*	*N*100
311ms	376ms	84ms

Tabela 6.15: Latências das componentes referentes ao *P*3*a*, *P*3*b* e *N*100 (Doente 1).

A componente 7, que pode representar o *P*3*b*, tem uma latência igual à componente 13 quando é utilizado a totalidade das componentes. Esta latência situa-se sensivelmente a meio das latências dos dois picos identificados na figura 6.19(b) como possíveis representantes do *P*3*b*. Os mapas topográficos das

componentes apresentadas na figura 6.21(b) e suas localizações usando o modelo esférico e modelo realista padrão são mostrados na figura 6.22.

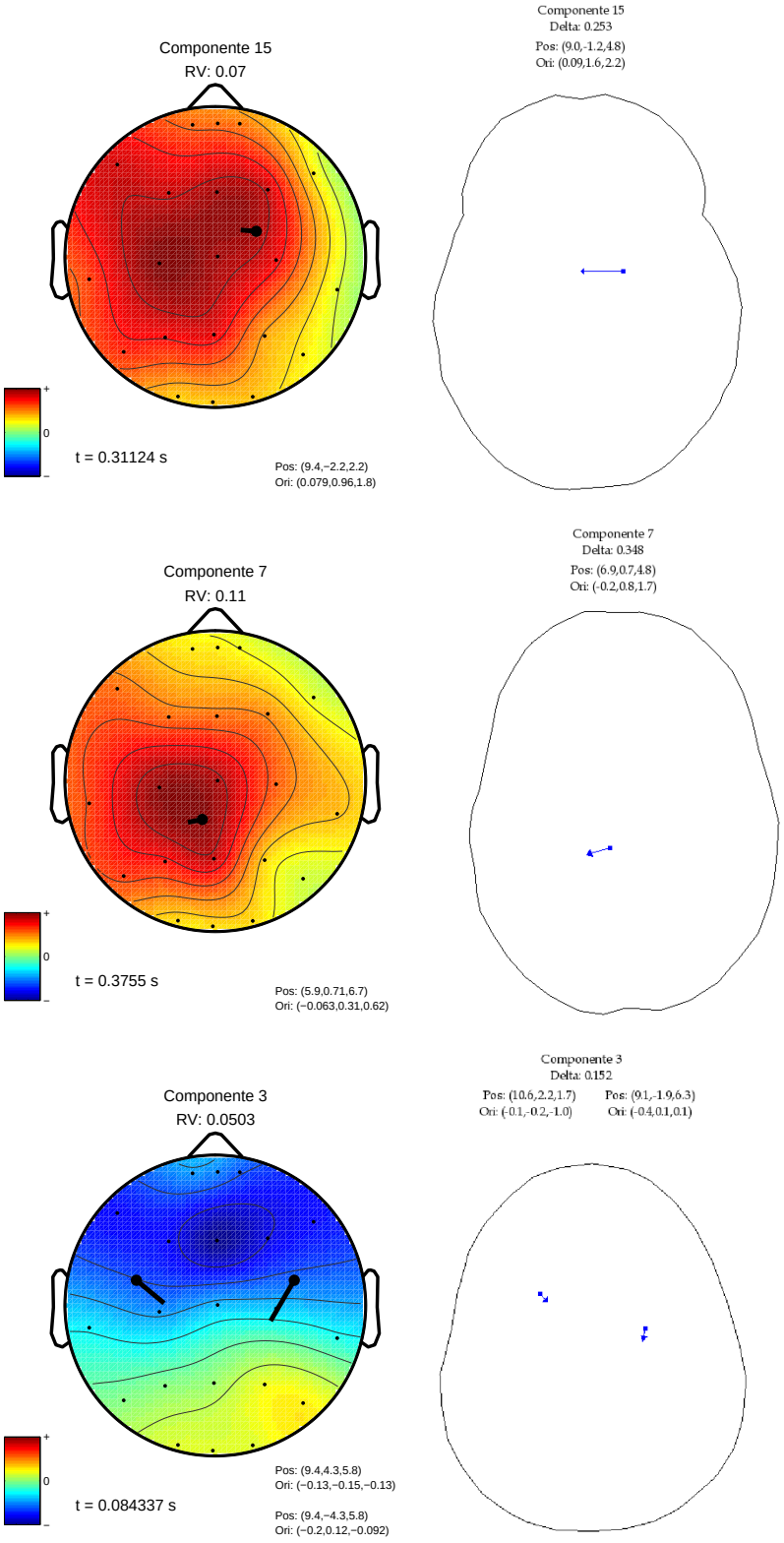


Figura 6.22: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Doente 1).

Componente 15	
Latência: 311**ms**	
Modelo Esférico	
RV: 0.07	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.4,-2.2,2.2)	(0.08,0.96,1.8)
Modelo Realista	
Delta (Δ): 0.253	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.0,-1.2,4.8)	(0.09,1.6,2.2)

Componente 7	
Latência: 376**ms**	
Modelo Esférico	
RV: 0.11	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(5.9,0.71,6.7))	(-0.063,0.31,0.62)
Modelo Realista	
Delta (Δ): 0.348	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.9,0.7,4.8)	(-0.2,0.8,1.7)

Componente 3	
Latência: 84**ms**	
ID: *N*100	
Modelo Esférico	
RV: 0.0503	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.4,4.3,5.8)	(-0.13,-0.15,-0.13)
(9.4,-4.3,5.8)	(-0.2,0.12,-0.09)
Modelo Realista	
Delta (Δ): 0.152	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(10.6,2.2,1.7)	(-0.1,-0.2,-1.0)
(9.1,-1.9,6.3)	(-0.4,0.1,0.1)

Tabela 6.16: Características das componentes escolhidas (Doente 1).

Analisando a figura 6.22 e a tabela 6.16 pode-se verificar que:

- A componente 15 poderá representar o $P3a$ devido à latência do pico. A actividade eléctrica positiva está concentrada na zona frontal esquerda (Fz). Quanto à localização do dipolo, este está situado na zona frontal (Fz) ligeiramente lateralizado para a direita, com orientação para a zona temporal esquerda em ambos os modelos.

- A componente 7 poderá representar o $P3b$ devido à latência do pico. A actividade eléctrica positiva está concentrada na zona parietal (Pz) esquerda. Quanto à localização do dipolo, este está situado na zona parietal (Pz) (característico do $P3b$), orientado para a zona temporal esquerda.

Quanto à localização dos dipolos, a componente 15 encontra-se lateralizada para o lado direito (característico de uma epilepsia temporal direita), mas, o dipolo da componente 7 está bastante centralizado, não se podendo concluir em relação a esta componente a localização do lado da epilepsia. As assimetrias das orientações destas duas componentes para o lado esquerdo vêm reforçar a ideia da presença de uma epilepsia temporal direita.

Analisando a figura 6.22 e com latência de 84ms, a componente 3 poderá representar o $N100$ (que não foi possível identificar na análise do envelope de potenciais originais). Esta componente tem uma actividade eléctrica negativa concentrada na zona frontal (Fz) e central (Cz) o que é também característico do $N100$. Os dipolos estão situados numa zona central lateralizados para a esquerda e direita, orientados para a zona central em ambos os modelos.

De notar que o *overlearning*, como foi demonstrado, aparecia aquando a aplicação da ACI para a totalidade das componentes, já não aparece para o potencial $P3b$ depois da redução do número de componentes a procurar pela ACI.

6.4.2 Doente 2

O doente 2, que tem uma epilepsia temporal esquerda, foi processado de forma análoga à do doente 1 e foram obtidos os resultados que em seguida se expõem.

6.4.2.1 Identificação do lado da epilepsia

A componente *P*300 original para este caso tem a seguinte distribuição apresentada na figura 6.23:

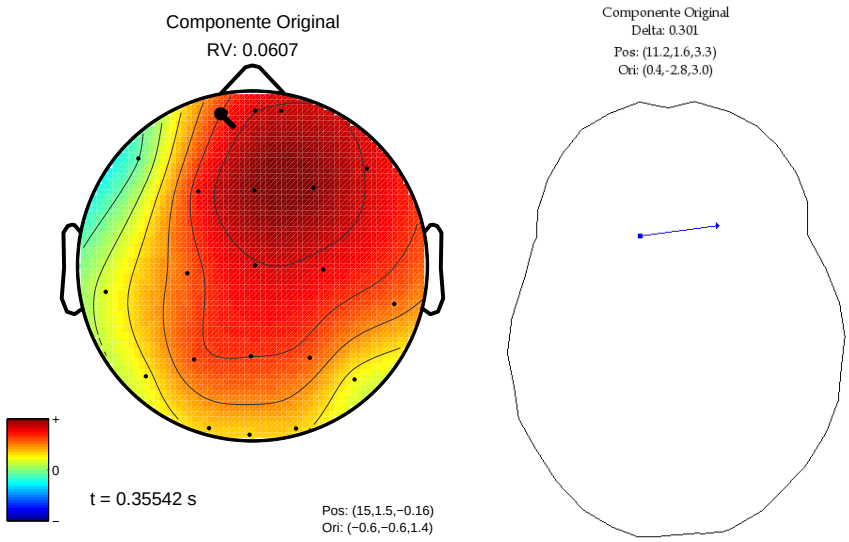


Figura 6.23: Mapa topográfico da componente P300 original e localização do dipolo (modelo esférico e realista) (Doente 2).

Componente Original	
Latência: 355**ms**	
ID: *P*300	
Modelo Esférico	
RV: 0.0607	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(15,1.5,-0.16)	(-0.6,-0.6,1.4)
Modelo Realista	
Delta (Δ): 0.301	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(11.2,1.6,3.3)	(0.4,-2.8,3.0)

Tabela 6.17: Características da componente original (Doente 2).

Observando a localização do dipolo, este encontra-se numa zona frontal ligeiramente para a esquerda, não se podendo dizer muita coisa sobre o lado da epilepsia. No entanto, a sua forte orientação para a zona temporal direita (modelo realista) dá indícios da presença de uma epilepsia temporal esquerda.

6.4.2.2 Identificação dos potenciais através da latência nos sinais originais

Analisando o resultado da aplicação do método de envelope dos potenciais, as latências onde provavelmente se encontram os potenciais em estudo estão presentes na tabela 6.18.

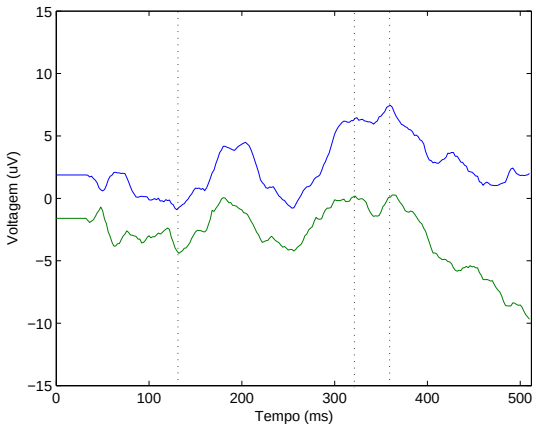


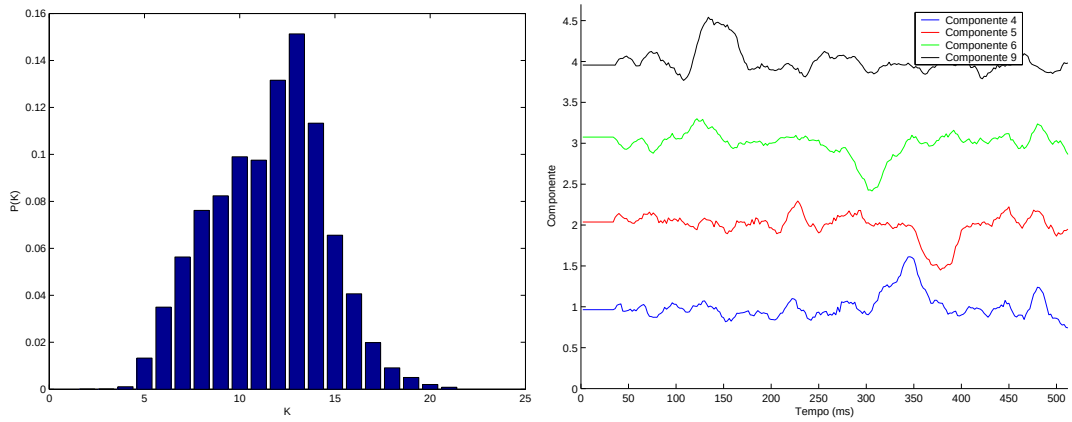
Figura 6.24: Envelope dos potenciais (Doente 2).

Potenciais	*P*3*a*	*P*3*b*	*N*100
Latências	326ms	361ms	131ms
Envelope	Máximos	Máximos	Mínimos

Tabela 6.18: Latências de cada um dos potenciais (Doente 2).

6.4.2.3 Identificação através das componentes calculadas pela ACI

Após a aplicação do critério BIC, como se pode ver na figura 6.25(a), o número mais provável de 13 fontes. Então, usando o algoritmo de ACI na procura deste número de componentes, foram escolhidas as componentes apresentadas na figura 6.25(b) como possíveis representantes dos potenciais em estudo.



(a) (b)

Figura 6.25: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 13 componentes (Doente 2).

Componente 6	Componente 4	Componente 5	Componente 9
*P*3*a*	*P*3*b*1	*P*3*b*2	*N*100
305ms	345ms	377ms	132ms

Tabela 6.19: Latências das componentes referentes ao *P*3*a*, *P*3*b* e *N*100 (classificação através da latência) (Doente 2).

Os mapas topográficos destas componentes e as suas localizações usando o modelo esférico e o modelo realista padrão são mostradas nas figuras 6.26 e 6.27.

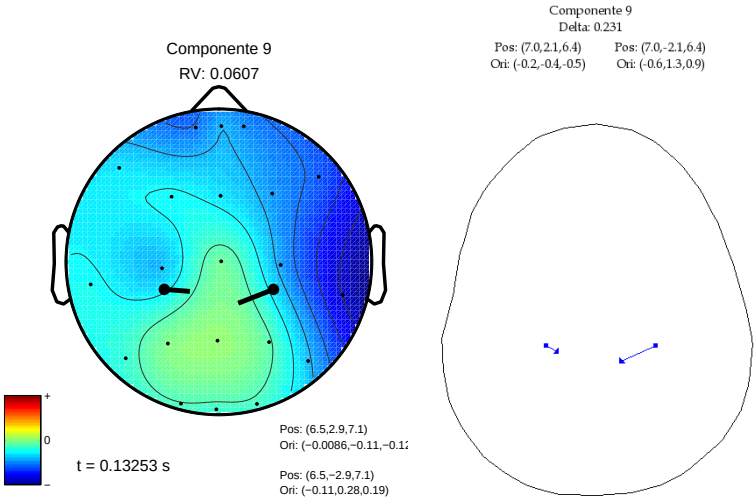


Figura 6.26: Mapas topográficos e localização das componentes escolhidas para o *N*100 após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Doente 2).

Componente 9	
Latência: 132**ms**	
ID: *N*100	
Modelo Esférico	
RV: 0.0607	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.5,2.9,7.1)	(-0.0086,-0.11,-0.12)
(6.5,-2.9,7.1)	(-0.11,0.28,0.19)
Modelo Realista	
Delta (Δ): 0.231	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.0,2.1,6.4)	(-0.2,-0.4,-0.5)
(7.0,-2.1,6.4)	(-0.6,1.3,0.9)

Tabela 6.20: Características das componentes escolhidas (Doente 2).

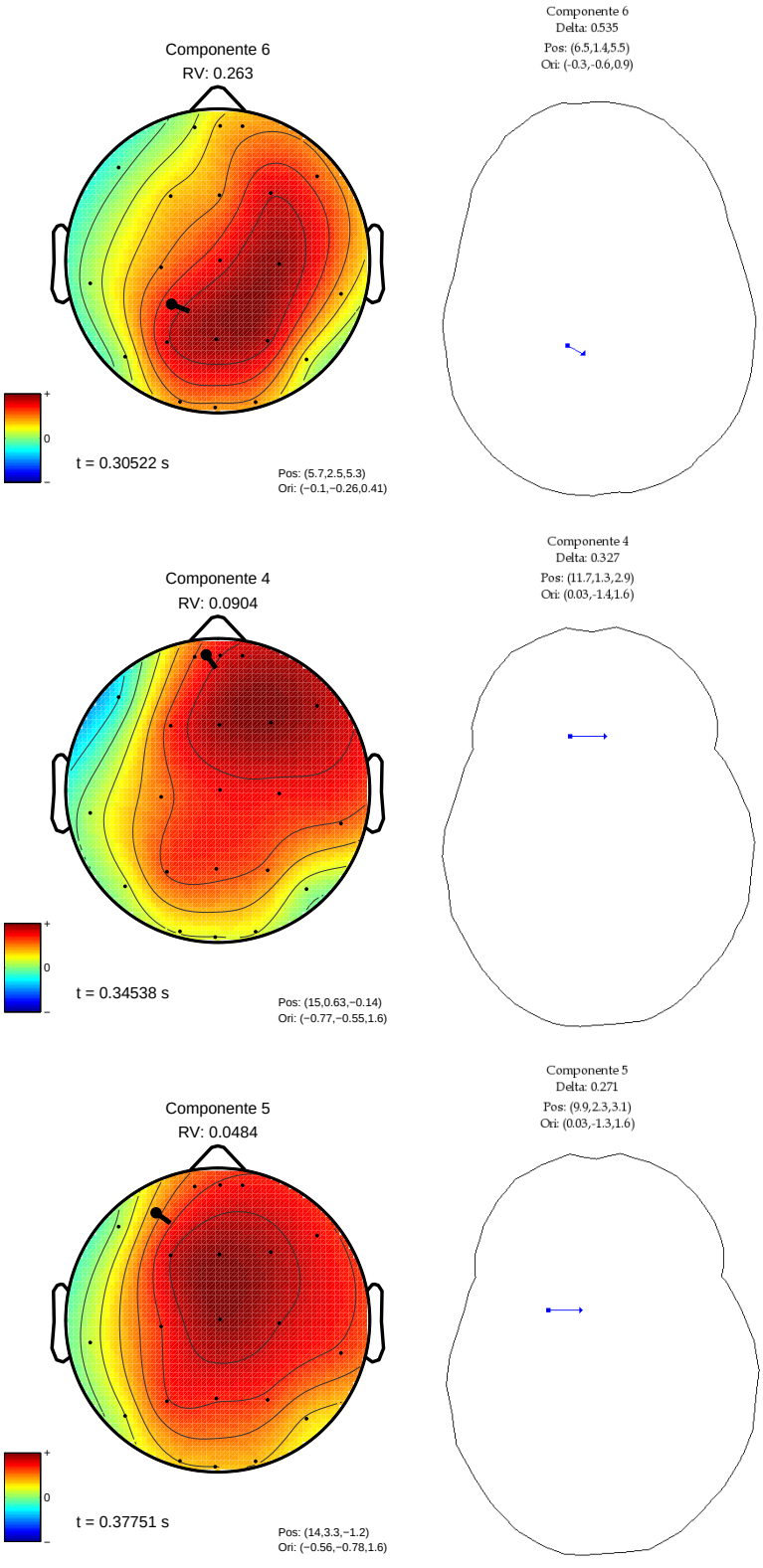


Figura 6.27: Mapas topográficos e localização das componentes escolhidas para o *P3a* e *P3b* após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Doente 2).

Componente 6	
Latência: 305**ms**	
Modelo Esférico	
RV: 0.263	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(5.7,2.5,5.3)	(-0.1,-0.26,0.41)
Modelo Realista	
Delta (Δ): 0.535	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.5,1.4,5.5)	(-0.3,-0.6,0.9)

Componente 4	
Latência: 345**ms**	
Modelo Esférico	
RV: 0.0904	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(15,0.63,-0.14)	(-0.77,-0.55,1.6)
Modelo Realista	
Delta (Δ): 0.327	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(11.7,1.3,2.9)	(0.03,-1.4,1.6)

Componente 5	
Latência: 377**ms**	
Modelo Esférico	
RV: 0.0484	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(14,3.3,-1.2)	(-0.56,-0.78,1.6)
Modelo Realista	
Delta (Δ): 0.271	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.9,2.3,3.1)	(0.03,-1.3,1.6)

Tabela 6.21: Características das componentes escolhidas (Doente 2).

Analisando a figura 6.27 e a tabela 6.21 pode-se verificar que:

- A componente 6 poderá representar o $P3a$ devido à latência do pico. A actividade eléctrica está concentrada na zona parietal (Cz) existindo também actividade eléctrica na zona frontal (Fz). Quanto à localização do dipolo, este está situado na zona parietal (Pz) esquerda tendo uma orientação para a zona temporal direita. Devido à distribuição eléctrica e ao mau valor do delta (péssimo acordo) quando é utilizado o modelo de dipolo único, para a localização das fontes, foi utilizado o modelo de dipolo duplo. O resultado é o apresentado na figura 6.28.

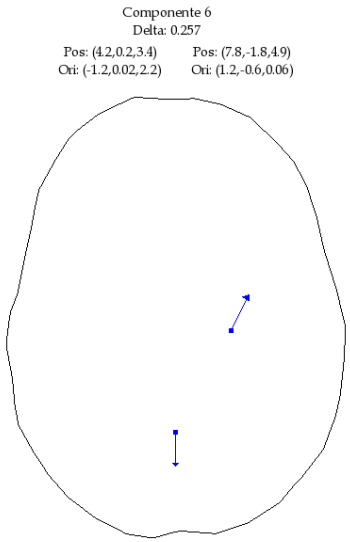


Figura 6.28: Localização utilizando o modelo de dipolo duplo (Doente 2).

Componente 6	
Modelo Realista	
Delta (Δ): 0.257	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(4.2,0.2,3.4)	(-1.2,0.02,2.2)
(7.8,-1.8,4.9)	(1.2,-0.6,0.06)

Tabela 6.22: Características da componente 6 (Doente 2).

O valor de desvio obtido demonstra que o modelo de dipolo duplo adapta-se bastante melhor a esta componente. Um dos dipolos está situado na zona central direita com orientação para a zona frontal direita. O outro dipolo encontra-se na zona parietal (característico do potencial $P3b$), orientado para a zona occipital.

- A componente 4 poderá representar o $P3a$ devido à latência do pico. A actividade eléctrica está concentrada na zona frontal direita(Fz). Quanto à localização do dipolo, este está situado na zona frontal (Fz) (característico do $P3a$), com clara orientação para a zona temporal direita.

- A componente 5 poderá representar o $P3b$ devido à latência do pico. A actividade eléctrica está concentrada na zona centro-frontal. Quanto à localização do dipolo, este está situado na zona frontal (Fz) mais à esquerda que a componente 4, possuindo uma orientação para a zona temporal direita.

Pelas topografias e pelas localizações dos dipolos, as componentes independentes identificadas que constituem o $P300$ não parecem representar o $P3a$ e o $P3b$ embora as suas latências apontem nesse sentido. Os dipolos não estão situados em locais característicos destes potenciais ($P3a$ mais posterior e $P3b$ mais anterior) indicando a possibilidade dos circuitos envolvidos na produção destes potenciais sofrerem alterações na presença de epilepsia focal temporal.

Parece continuar a existir *overlearning* nas componentes 4 e 5, mas, as distribuições eléctricas diferem um pouco e sobretudo o dipolo da componente 5 encontra-se bastante mais lateralizado para a esquerda, numa zona mais central, denotando a possibilidade da ocorrência de um processo cerebral diferente na componente 4. Aliás, o maior número de componentes pode explicar a grande duração do potencial $P300$ neste caso. Estas componentes permitem identificar (através da localização das fontes neuronais (componente 5) e suas orientações para a direita (ambas as componentes)) com maior rigor o lado da epilepsia temporal, neste caso esquerda.

Observando a figura 6.26, verifica-se que a componente 9 tem o pico com latência de 132ms, característico do $N100$. Esta componente tem uma actividade eléctrica concentrada na zona frontal (Fz) e zonas temporais direita e esquerda. Os dipolos estão situados na zona central lateralizados para a esquerda e direita, com orientações para a zona parietal.

6.4.3 Doente 3

Os resultados para o doente 3, que tem uma epilepsia temporal esquerda, foram processados usando a mesma metodologia que nos casos anteriores. Comece-se, pois pela identificação do lado da epilepsia usando o potencial original.

6.4.3.1 Identificação do lado da epilepsia

Neste caso, a componente *P*300 original tem a seguinte distribuição:

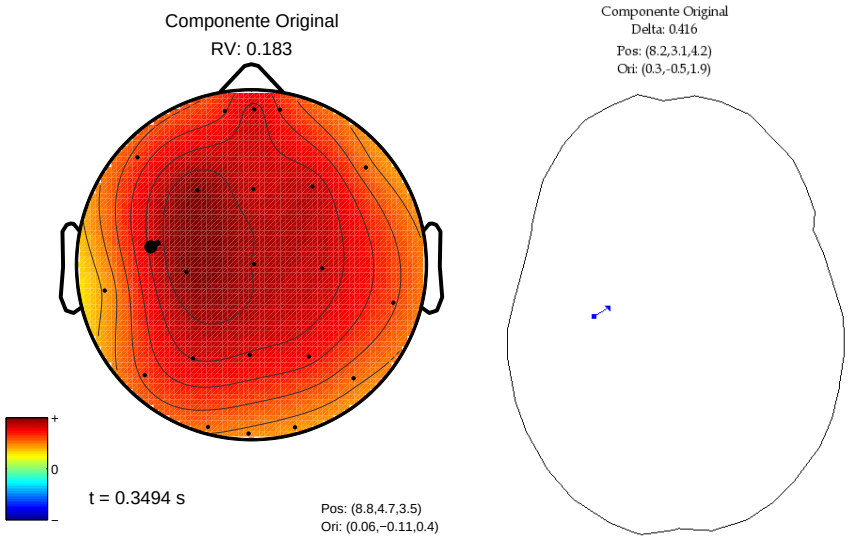


Figura 6.29: Mapa topográfico da componente P300 original e localização do dipolo (modelo esférico e realista) (Doente 3).

Componente Original	
Latência: 349**ms**	
ID: *P*300	
Modelo Esférico	
RV: 0.183	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.8,4.7,3.5)	(0.06,-0.11,0.4)
Modelo Realista	
Delta (Δ): 0.416	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.2,3.1,4.2)	(0.3,-0.5,1.9)

Tabela 6.23: Características da componente original (Doente 3).

Observando a localização do dipolo e a sua orientação, pode-se dizer que o indivíduo tem uma epilepsia temporal esquerda.

6.4.3.2 Identificação dos potenciais através da latência nos sinais originais

Analisando o resultado da aplicação do método de envelope dos potenciais, as latências onde provavelmente se encontram os potenciais em estudo estão presentes na tabela 6.24.

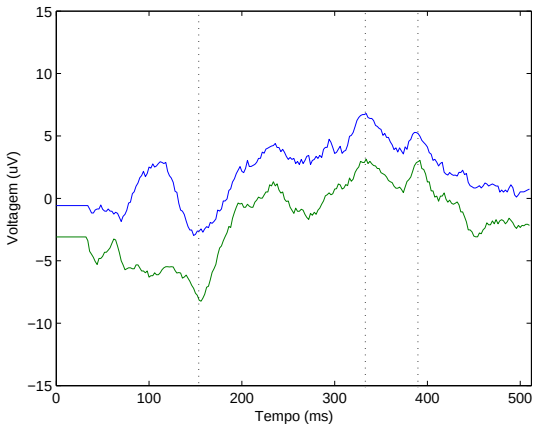


Figura 6.30: Envelope dos potenciais (Doente 3).

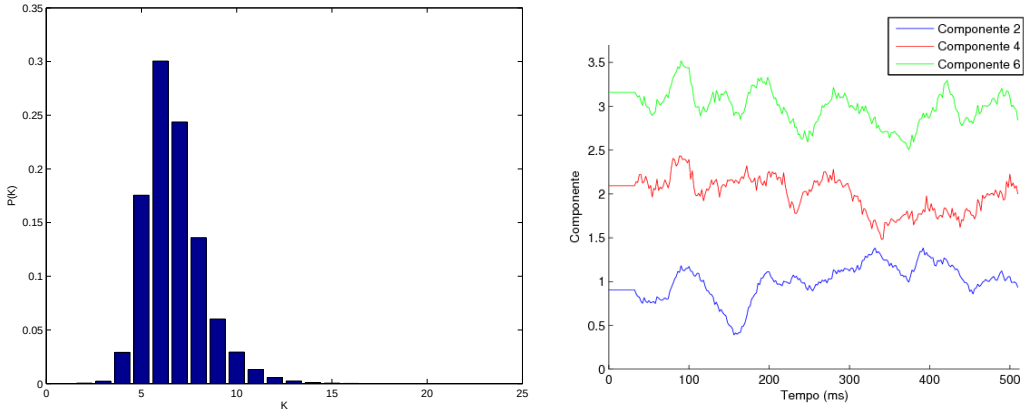
Potenciais	*P*3*a*	*P*3*b*	*N*100
Latências	334ms	388ms	150ms
Envelope	Máximos	Máximos	Mínimos

Tabela 6.24: Latências de cada um dos potenciais (Doente 3).

Denotar que neste caso, a latência do *N*100, que está relacionada com respostas físicas imediatas, está bastante atrasada, o que implica também um grande atraso nos potenciais *P*3*a* e *P*3*b*.

6.4.3.3 Identificação através das componentes calculadas pela ACI

Utilizando o critério BIC, como se pode ver na figura 6.31(a), o número mais provável de fontes presentes nos dados é seis. Utilizando o algoritmo de ACI na procura deste número de componentes, foram escolhidas as componentes apresentadas na figura 6.31(b) como possíveis representantes dos potenciais *P*3*a*, *P*3*b* e *N*100.



(a) (b)

Figura 6.31: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 6 componentes (Doente 3).

Componente 2	Componente 4	Componente 6
*P*3*a*	*P*3*b*	*N*100
340ms	376ms	160ms

Tabela 6.25: Latências das componentes referentes ao *P*3*a*, *P*3*b* e *N*100 (classificação através da latência) (Doente 3).

Os mapas topográficos destas componentes e suas localizações usando o modelo esférico e modelo realista padrão são mostrados na figuras 6.32.

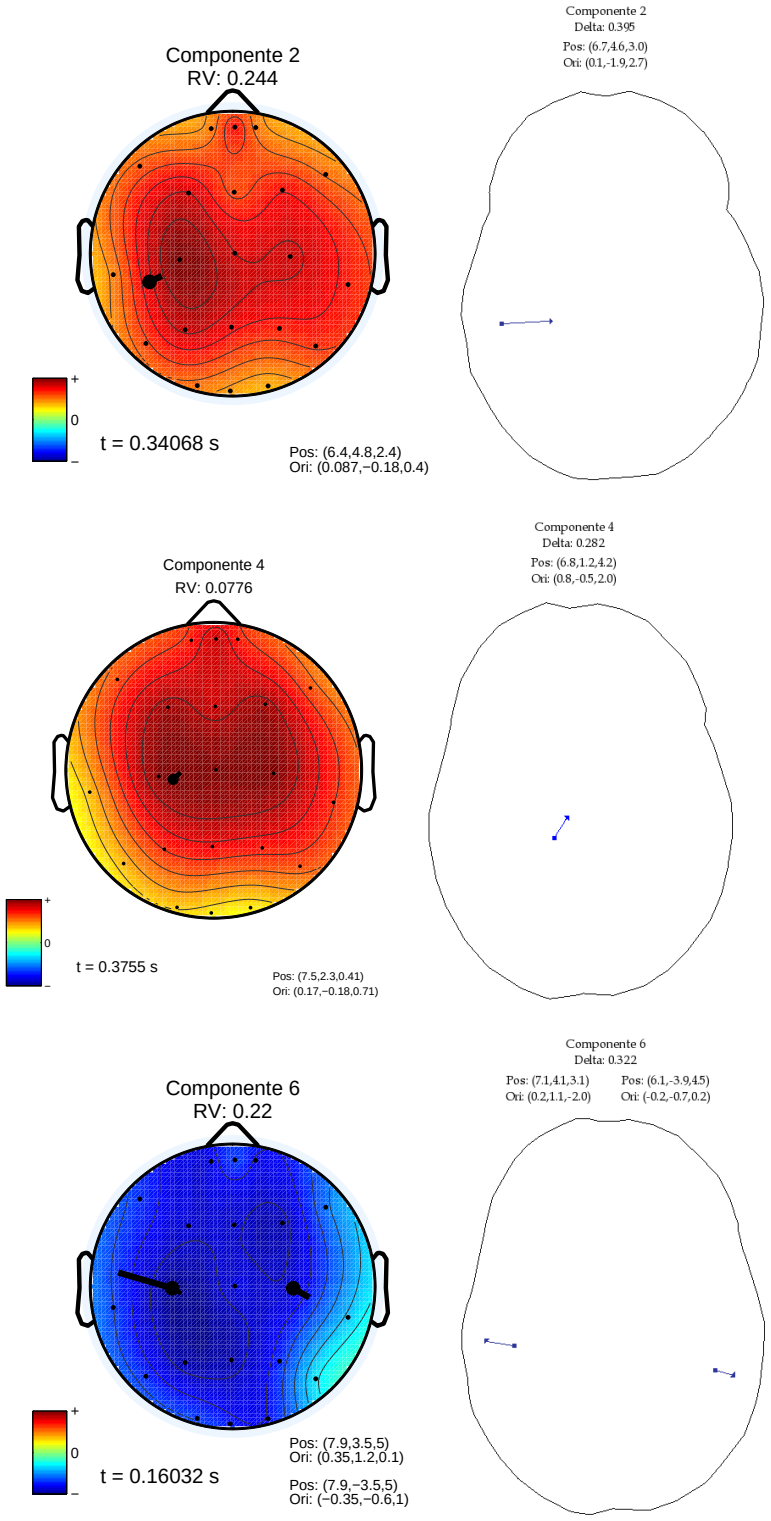


Figura 6.32: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Doente 3).

Componente 2	
Latência: 340**ms**	
Modelo Esférico	
RV: 0.244	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.4,4.8,2.4)	(0.09,-0.18,0.4)
Modelo Realista	
Delta (Δ): 0.395	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.7,4.6,3.0)	(0.1,-1.9,2.7)

Componente 4	
Latência: 376**ms**	
Modelo Esférico	
RV: 0.0776	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.5,2.3,0.41))	(0.17,-0.18,0.71)
Modelo Realista	
Delta (Δ): 0.282	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(6.8,1.2,4.2)	(0.8,-0.5,2.0)

Componente 6	
Latência: 160**ms**	
ID: *N*100	
Modelo Esférico	
RV: 0.22	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.9,3.5,5)	(-0.23,0.05,0.01)
(7.9,-3.5,5)	(-0.23,-0.19,-0.33)
Modelo Realista	
Delta (Δ): 0.322	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.1,4.1,3.1)	(0.2,1.1,-2.0)
(6.1,-3.9,4.5)	(-0.2,-0.7,0.2)

Tabela 6.26: Características das componentes escolhidas (Doente 3).

Analisando a figura 6.32 e a tabela 6.26 pode-se verificar que:

- A componente 2 poderá representar o $P3a$ devido à latência do pico, mas, a actividade eléctrica está concentrada na zona central (Cz), que é característico do $P3b$. Quanto à localização do dipolo, este está situado na zona parietal bastante lateralizado para a esquerda, com orientação para a zona temporal direita.

- A componente 4 poderá representar o $P3b$ devido à latência do pico e devido à actividade eléctrica estar concentrada na zona central (Cz). Quanto à localização do dipolo, este está situado na zona central (Cz) esquerda com orientação para a zona frontal direita. Apesar da latência desta componente, o dipolo encontra-se numa região anterior ao dipolo da componente 2, ao contrário do que se esperava.

A localização dos dipolos (lateralizados para a esquerda) e suas orientações para o lado direito das componentes 2 e 4 vêm reforçar a ideia da presença de uma epilepsia temporal esquerda que havia sido dada pela componente original.

Pela figura 6.32, com latência de 160ms e com actividade eléctrica concentrada na zona central (Cz), a componente 6 poderá representar o $N100$. Os dipolos estão situados na zona centro-parietal bastante lateralizados para a esquerda e direita, com o dipolo esquerdo orientado para a zona temporal esquerda e o dipolo direito orientado para a zona temporal direita, isto em ambos os modelos.

6.4.4 Doente 4

O quarto e último caso estudado (doente com epilepsia temporal direita) foi processado segundo o método descrito e obteve-se os resultados que seguidamente se apresentam.

6.4.4.1 Identificação do lado da epilepsia

A componente *P*300 original para este caso tem a distribuição apresentada na figura 6.33:

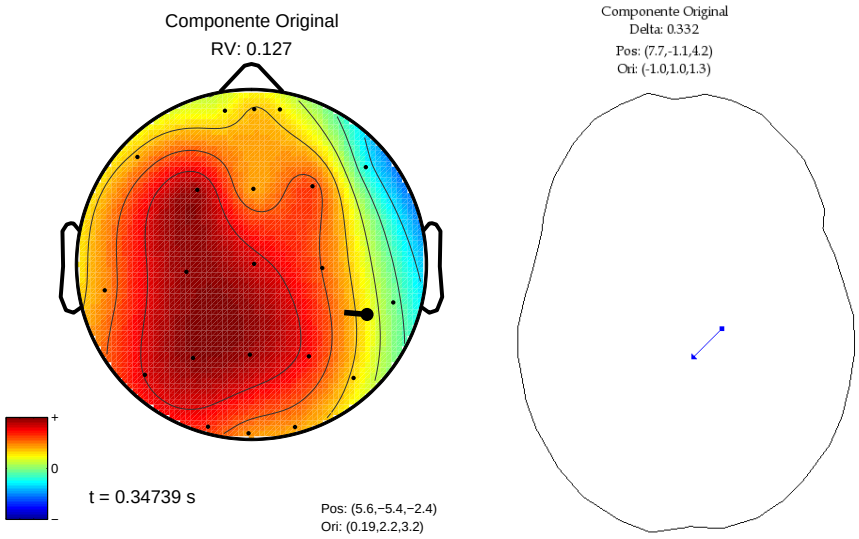


Figura 6.33: Mapa topográfico da componente P300 original e localização do dipolo (modelo esférico e realista) (Doente 4).

Componente Original	
Latência: 347**ms**	
ID: *P*300	
Modelo Esférico	
RV: 0.127	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(5.6,-5.4,-2.4)	(0.19,2.2,3.2)
Modelo Realista	
Delta (Δ): 0.332	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.7,-1.1,4.2)	(-1.0,1.0,1.3)

Tabela 6.27: Características da componente original (Doente 4).

Observando a localização do dipolo e a sua orientação, pode-se dizer que o indivíduo tem uma epilepsia temporal direita.

6.4.4.2 Identificação dos potenciais através da latência nos sinais originais

Aplicando o método de envelope dos potenciais a este caso, as latências identificadas onde provavelmente se encontram os potenciais *P*3*a*, *P*3*b* e *N*100 estão presentes na tabela 6.28.

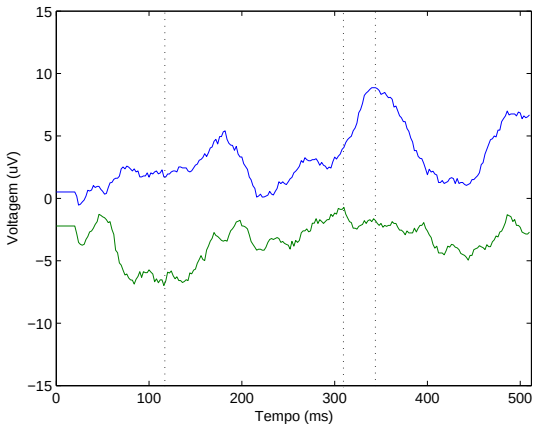


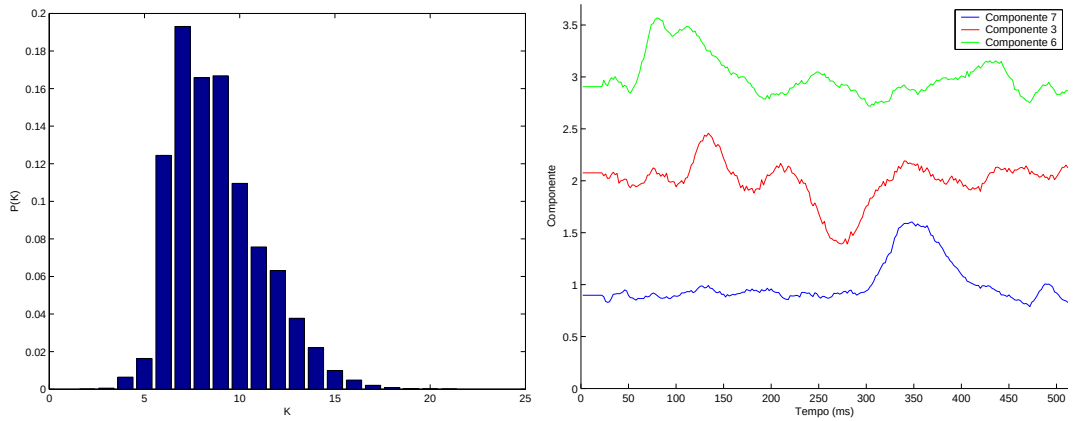
Figura 6.34: Envelope dos potenciais (Doente 4).

Potenciais	*P*3*a*	*P*3*b*	*N*100
Latências	308ms	345ms	118ms
Envelope	Mínimos	Máximos	Mínimos

Tabela 6.28: Latências de cada um dos potenciais (Doente 4).

6.4.4.3 Identificação através das componentes calculadas pela ACI

Utilizando o critério BIC, verificou-se que para este caso, o número mais provável de fontes presentes nos dados originais é de sete (ver figura 6.35(a)). Aplicando o algoritmo de ACI procurando este número de componentes, foram escolhidas as componentes apresentadas na figura 6.35(b) como possíveis representantes dos potenciais em estudo.



(a) (b)

Figura 6.35: a) Probabilidade de adequação de cada modelo de ACI K (que denota o número de componentes) aos dados originais. b) Componentes independentes escolhidas após cálculo da ACI para 7 componentes (Doente 4).

Componente 3	Componente 7	Componente 6
$P3a$	$P3b$	$N100$
280ms	347ms	78ms

Tabela 6.29: Latências das componentes referentes ao $P3a$, $P3b$ e $N100$ (classificação através da latência) (Doente 4).

As projecções destas componentes e localizações das fontes neuronais para o modelo esférico e realista padrão são mostradas na figura 6.36.

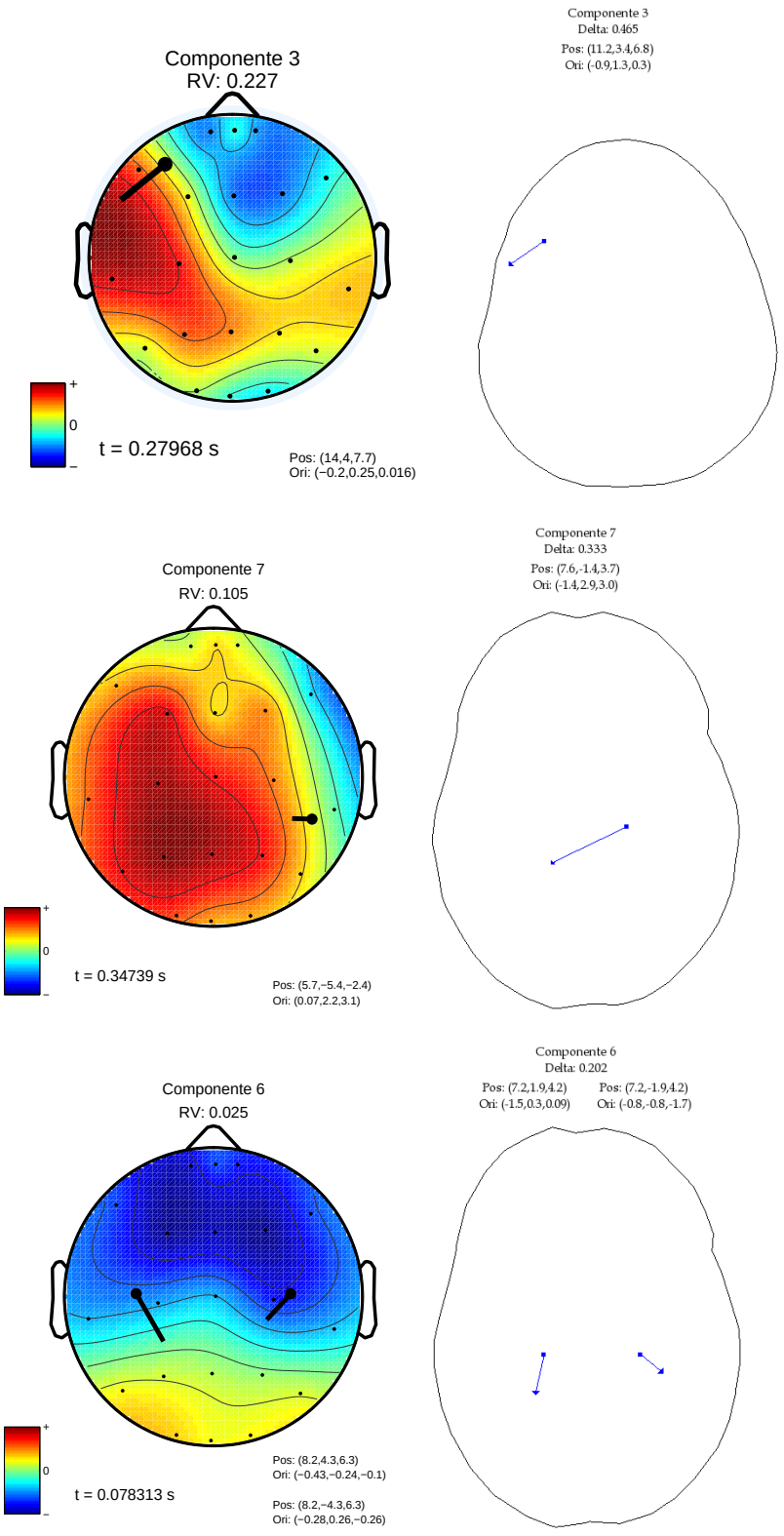


Figura 6.36: Mapas topográficos e localização das componentes escolhidas após redução (lado esquerdo - modelo esférico, lado direito - modelo realista) (Doente 4).

Componente 3	
Latência: 280**ms**	
Modelo Esférico	
RV: 0.227	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(14,4.0,7.7)	(-0.2,0.25,0.02)
Modelo Realista	
Delta (Δ): 0.465	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(11.2,3.4,6.8)	(-0.9,1.3,0.3)

Componente 7	
Latência: 347**ms**	
Modelo Esférico	
RV: 0.105	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(5.7,-5.4,-2.4)	(0.07,2.2,3.1)
Modelo Realista	
Delta (Δ): 0.333	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.6,-1.4,3.7)	(-1.4,2.9,3.0)

Componente 6	
Latência: 78**ms**	
ID: *N*100	
Modelo Esférico	
RV: 0.025	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(8.2,4.3,6.3)	(-0.43,-0.24,-0.1)
(8.2,-4.3,6.3)	(-0.28,0.26,-0.26)
Modelo Realista	
Delta (Δ): 0.202	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(7.2,1.9,4.2	(-1.5,0.3,0.09)
(7.2,-1.9,4.2)	(-0.8,-0.8,-1.7)

Tabela 6.30: Características das componentes escolhidas (Doente 4).

Analisando a figura 6.36 e a tabela 6.30 pode-se verificar que:

- A componente 3 poderá representar o $P3a$ devido à latência do pico. A actividade eléctrica está concentrada na zona frontal (Fz) e temporal esquerda. Existe também actividade eléctrica na zona parietal (Pz). O dipolo está situado na zona frontal lateralizado para a esquerda. Embora a localização do dipolo dê uma errada indicação do lado da epilepsia, a sua forte orientação para a zona temporal esquerda denota a presença de uma epilepsia temporal direita.

  Devido à destribuição eléctrica bastante não-uniforme e ao mau valor do delta (péssimo acordo) do modelo de dipolo único, para a localização das fontes, foi utilizado o modelo de dipolo duplo. O resultado é apresentado na figura 6.37.

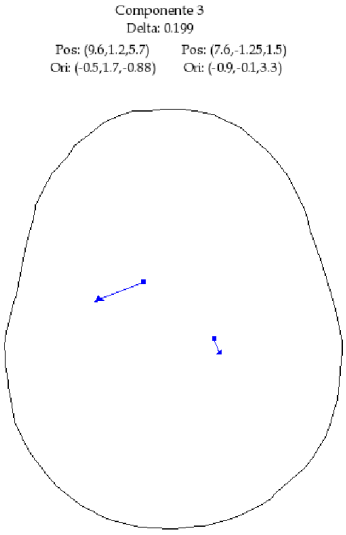


Figura 6.37: Localização utilizando o modelo de dipolo duplo (Doente 4).

Componente 3	
Modelo Realista	
Delta (Δ): 0.199	
Posição (x,y,z)(cm)	**Orientação (x,y,z)(μA.m)**
(9.6,1.2,5.7)	(-0.5,1.7,-0.88)
(7.6,-1.25,1.5)	(-0.9,-0.1,3.3)

Tabela 6.31: Características da componente 3 (Doente 4).

O valor de desvio obtido demonstra que o modelo de dipolo duplo se adapta bastante melhor a esta componente. Um dos dipolos está situado na zona centro-frontal (característico do potencial $P3a$) com orientação para a zona temporal esquerda. Com uma localização mais central, a orientação para a zona temporal esquerda indica a presença de uma epilepsia temporal direita. O outro dipolo encontra-se numa zona centro parietal (característico do potencial $P3b$) orientado para a zona parietal.

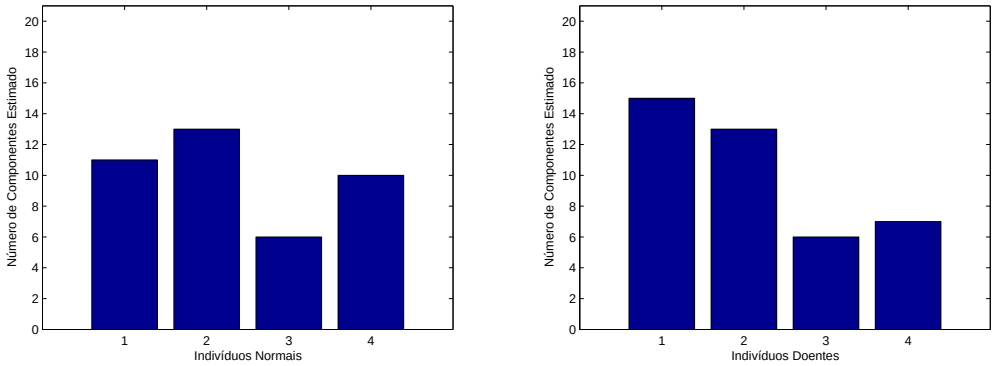
- A componente 7 poderá representar o $P3b$ devido à latência do pico e devido à actividade eléctrica concentrada na zona parietal esquerda (Pz). Quanto à localização do dipolo, este está situado na zona parietal direita, com orientação para a zona temporal esquerda.

A localização do dipolo e a orientação para o lado esquerdo da componente 7 como também a forte orientação para a zona temporal esquerda da componente 3 (caso do dipolo único) vêm reforçar a ideia da presença de uma epilepsia temporal direita.

Analisando a figura 6.36, a componente 6 poderá representar o $N100$ por esta apresentar uma latência do pico de 78ms. Esta componente apresenta uma actividade eléctrica concentrada nas zonas frontal (Fz) e temporal direita e esquerda. Os dipolos estão situados na zona central lateralizados para a esquerda e direita e orientados para a zona parietal.

6.5 Número de Componentes

O número estimado de componentes presentes em cada registo calculado através do método BIC/MDL é apresentado na figura 6.38.



(a) Indivíduos Normais (b) Indivíduos Doentes

Figura 6.38: Número estimado de componentes presentes em cada um dos registos estudados.

Como se pode verificar na figura 6.38, a média do número de componentes presentes nos registos é semelhante tanto no caso dos indivíduos normais como nos doentes.

Capítulo 7: Módulo pyICA

7.1 Introdução

No decorrer deste trabalho verificou-se que as ferramentas existentes eram todas programadas em Matlab. Para facilitar a implementação de novas técnicas de ACI noutras linguagens e possibilitar a sua utilização sem requerer a compra de software como o Matlab, foi decidido criar uma biblioteca livre e *Open-Source* que implementasse as estruturas básicas dos algoritmos de ACI. Estas estruturas foram criadas através de classes e foram implementadas duas técnicas de ACI, o Fastica e o IcaML. Foi também implementada a ACP como também o método BIC para o cálculo do número de componentes. As linguagens utilizadas foram o Python e Numeric a servirem de interface para uma biblioteca de C/C++. A decisão da implementação em C++ deve-se ao facto de que na utilização de grandes estruturas de dados e de estruturas repetitivas (como é o caso da ACI), as linguagens interpretadas como o Python e o próprio Matlab (embora ofereçam métodos rápidos e fáceis de programação), não têm um desempenho muito rápido e eficiente, o que pode prolongar muito o cálculo da ACI se o número de amostras ou o número de misturas for muito elevado. De referir que estas ferramentas desenvolvidas não foram utilizadas nas análise anteriormente apresentadas.

Em seguida são descritas as várias ferramentas utilizadas no desenvolvimento da biblioteca proposta.

7.2 Python

A linguagem de programação Python é uma linguagem interpretada, interactiva e completamente orientada por objectos. Esta linguagem é muitas vezes comparada ao Perl, Scheme ou Java.

Esta linguagem tem uma sintaxe clara e suporta módulos, classes, excepções e tipos de dados de alto nível. Existem muitas interfaces para diversas bibliotecas como também para vários sistemas de janelas (GUI) como o X11, Motif, Tk, Mac, wxWidgets.

Esta linguagem permite também uma fácil adição de módulos escritos em C/C++.

A linguagem Python é livre e portável, funcionando em sistemas Linux, UNIX, Windows, OS/2, Mac, Amiga e muitos outros. A versão utilizada no desenvolvimento desta biblioteca é a 2.4 e encontra-se disponível em [7].

7.3 Python Numeric

Um dos módulos existentes para Python é o Python Numeric (Numpy). O Numpy implementa estruturas de dados e funções que permitem o processamento de matrizes N-dimensionais. A utilização destas estruturas como interface para a biblioteca desenvolvida é vantajosa. Uma das vantagens é a de tornar automaticamente o pyIca como extensão de outros módulos como o MatPlotLib, o SciPy, o PyPlot, etc. Estes módulos permitem posterior processamento com outras técnicas ou visualizações gráficas dos dados processados e dos resultados obtidos através do pyIca. O módulo Numpy encontra-se disponível em [6].

7.4 IT++

Para a programação da biblioteca em C++ foi utilizada como base a biblioteca IT++. A IT++ é uma biblioteca de classes e funções de matemática, processamento de sinal, processamento de voz e comunicações. Esta biblioteca tem sido desenvolvida por investigadores nestas áreas e tem sido muito utilizada por estes na industria das comunicações e nas universidades. O IT++ faz um extenso uso de bibliotecas *open-source* existentes como a FFTW, ATLAS e a LAPACK para aumento das suas funcionalidade, precisão e velocidade. O IT++ funciona em Sun Solaris, Linux, Microsoft Windows e em Mac OS X. A versão utilizada foi a 3.8.1 e encontra-se disponível em [12].

7.5 cBLAS

Uma das operações mais morosas no algoritmo IcaML é a operação dada pela expressão 7.1:

$$A = x\,y^T + A, \tag{7.1}$$

onde A é uma matriz, x e y são dois vectores. Como a biblioteca IT++ não disponibiliza uma função optimizada que efectue esta operação de um modo eficiente, foi desenvolvida uma extensão do IT++ utilizando a biblioteca cBLAS.

A biblioteca cBLAS corresponde a um *port* para C da biblioteca BLAS. O BLAS (*Basic Linear Algebra Subprograms*) correponde a uma colecção de sub-programas em Fortran optimizados para operações de álgebra linear como as operações vector-vector, matriz-vector e matriz-matriz.

7.6 Swig

Para fazer o *port* da biblioteca C++ e fazer a interface para Python e o Numpy, foi utilizado o Swig. Swig é o *Simplified Wrapper Interface Generator* escrito por David Beazly [10]. O Swig utiliza um ficheiro de interface derivado de um ficheiro existente de *include* de C/C++. O Swig processa essa interface e gera código C/C++ que faz o interface de Python para C/C++. O resultado é um módulo de Python que pode ser importado em Python e onde se pode invocar classes, funções, métodos e estruturas de dados criadas em C/C++. A ferramenta Swig encontra-se disponível em [10].

7.7 Resultados

No momento a biblioteca foi desenvolvida em ambiente Linux. O método Fastica foi implementado para estruturas de classes tendo como base o código existente na biblioteca IT++. O método para a ACP foi implementado de raiz. Os métodos IcaML e BIC foram implementadas de raiz em C++ mas tendo por base o código Matlab, efectuando algumas alterações em algumas áreas para aumento de eficiência (poupanças de memória) e velocidade.

Em seguida vão ser comparados, em termos de velocidade, as implemenções dos métodos IcaML e BIC em diferentes linguagens de programação. Cada valor temporal (em segundos) colocado nas tabelas 7.1, 7.2 e 7.3 resulta da média de três repetições da mesma operação. Para efectuar estes testes foi utilizada uma plataforma Linux (distribuição Ubuntu 5.04) usando um computador equipado com um processador AMD Athlon 64 3500+ e com 1 Gb de memória RAM. As bibliotecas implementadas para C++ e para Python (pyICA) foram compiladas de forma a fazerem uso das instruções SSE2.

Misturas\Código	**Matlab**	**C++**	**PyIca**
21	7.5 s	3.4 s	3.9 s
32	32.5 s	14.2 s	14.8 s
64	2157.3 s	470.4 s	484.0 s

Tabela 7.1: Velocidade do algoritmo IcaML em diferentes ambientes para números de misturas diferentes com 256 instantes.

Misturas\Código	Matlab	C++	PyIca
512	5.8 s	3.1 s	3.6 s
1024	8.1 s	5.0 s	5.9 s

Tabela 7.2: Velocidade do algoritmo IcaML em diferentes ambientes para número de misturas fixo (21 misturas) e diferentes números de instantes.

Misturas\Código	Matlab	C++	PyIca
10	91.6 s	10.7 s	15.1 s
20	508.3 s	68.5 s	91.7 s
30	1279.4 s	200.4 s	254.7 s

Tabela 7.3: Velocidade do algoritmo BIC em diferentes ambientes para números de misturas diferentes com seis fontes em cada caso mantendo o número de amostras (256 instantes).

Pode-se verificar pelas tabelas 7.1, 7.2 e 7.2 que a biblioteca mais rápida é a que foi desenvolvida para C++ para todos os casos em estudo. Embora a biblioteca em C++ seja a mais rápida, esta é com certeza a mais difícil de utilizar. Para facilitar o seu uso, como já foi referido, foi feito um *port* desta biblioteca para uma linguagem interpretada (neste caso o Python). Esse *port* teve como resultado uma biblioteca chamada pyIca. Os resultados em termos de velocidade da biblioteca pyIca, apontam que esta biblioteca apresentar uma velocidade um pouco mais baixa do que a versão C++ devido ao *overhead* introduzido pelo Python. No entanto, a biblioteca pyIca continua a apresentar uma velocidade bastante maior do que a versão Matlab para todos os casos em estudo.

Capítulo 8: Conclusão

Como conclusão geral deste trabalho pode dizer-se que os objectivos propostos foram atingidos. O principal objectivo do projecto, que era a aplicação da ACI reduzindo o processamento a um número mínimo de componentes, foi conseguido através do critério BIC/MDL, para a escolha desse número. Foi também verificada a influência negativa provocada pela sub-divisão das componentes na aplicação da ACI no estudo do $P3a$, $P3b$ e $N100$, quando não se reduz o número de componentes a procurar.

A diminuição do número de componentes está associada, nos casos estudados, a uma redução no ruído e nos artefactos nas componentes calculadas, o que melhora a identificação das componentes de interesse. Outra das evidentes vantagens na utilização de um menor número de componentes é a maior velocidade na análise e na selecção das componentes de interesse. Esta questão pode ser muito importante quando são utilizados muitos eléctrodos para a recolha dos sinais (quanto maior o número de eléctrodos, maior o número de misturas e, consequentemente, maior será o tempo de processamento).

Relativamente a conclusões particulares é possível referir as seguintes:

- O algoritmo de ACI, especializado na separação de fontes com distribuição super-Gaussiana, permitiu, em todos os casos estudados, a separação do potencial $P300$ em várias componentes, e, mesmo após a redução do número de componentes, o $P300$ foi sempre decomposto em duas componentes (tendo como critério de selecção o valor das latências). O potencial $N100$ foi sempre identificado através de uma componente após a redução de componentes. Verificou-se que nalguns casos, sem redução de componentes, o potencial $N100$ era decomposto em duas componentes que apresentavam a mesma distribuição eléctrica no escalpe.

- Para os casos estudados, a ACI foi aplicada no cálculo de todas componentes e, posteriormente, no cálculo de apenas algumas componentes. Este procedimento foi feito para a verificação da possível perda de componentes que poderiam ser importantes neste estudo. Devido ao número reduzido de casos estudados, não se pode afirmar com certeza absoluta que nunca ocorra perda de informação importante, no entanto, verificou-se uma tendência para o problema de *overlearning* desaparecer, ou seja, duas ou mais componentes que representam a mesma actividade cerebral foram substituídas por apenas uma componente, como seria desejável, e em nenhum dos casos estudados houve perda de componentes importantes para o estudo do $P3a$, $P3b$ e $N100$.

- Como se pode verificar na figura 6.3, uma das dificuldades na utilização do modelo realista foi a estimativa da posição inicial. Esta dificuldade prende-se com o reduzido número de eléctrodos

utilizados nos registos, agravando-se quando é utilizado um modelo de duplo dipolo. Para facilitar o estudo utilizando um modelo de múltiplos dipolos, é necessário fazer os registos recorrendo a um maior número de eléctrodos.

- Nos indivíduos normais, os dipolos obtidos para as componentes que representam o *P*3*a* (critério da latência), estão situados numa zona média e numa posição anterior (zona frontal) relativamente aos dipolos para as componentes que representam o *P*3*b* (zona parietal). Em dois dos casos estudados (Normais) notou-se também que os potenciais *P*3*a* e *P*3*b* estão sobrepostos numa das componentes a qual, tendo em conta a sua latência, poderia representar apenas o *P*3*a*. Ou seja, apesar da latência das componentes, estas apresentavam também características pertencentes ao *P*3*b*, como sejam a topografia da actividade eléctrica e a posição do dipolo. Esta situação pode ser explicada pela grande variabilidade do *P*300. A latência e amplitude do *P*300 são influenciadas por muitos factores, como por exemplo, a idade do indivíduo, se houve ingestão recente de comida, a temperatura corporal, a luminosidade do ambiente, estado de espírito do indivíduo, personalidade do indivíduo, etc. Todas estas condicionantes influenciam a morfologia do *P*300 e por conseguinte as suas componentes *P*3*a* e *P*3*b*. Normalmente, a mais afectada deverá ser a *P*3*a* por esta estar primordialmente relacionada com os sistemas de alerta imediato e ser, por isso, mais curta e de maior habituação. A componente *P*3*b* é uma componente mais longa e está relacionada com o processamento da informação e com a actualização da memória e do estado de expectativa. É pois, bastante plausível, que, em alguns casos, haja sobreposição de ambas, resultando numa evidente dificuldade de separação. Esta impossibilidade na separação destas componentes (potenciais) poderá também dever-se à possível grande interligação dos circuitos neuronais subjacentes aos potenciais *P*3*a* e *P*3*b*, os quais se encontram intimamente dependente do tipo de estímulo utilizado para a obtenção dos registos.

- Nos indivíduos doentes, para as componentes que representam o *P*3*a* e *P*3*b* (critério da latência), os dipolos, na maioria dos casos, não estão situados em locais característicos destes potenciais indicando a possibilidade dos circuitos envolvidos na produção destes potenciais sofrerem algumas alterações na presença de epilepsia focal temporal. A localização e orientação dos dipolos nos indivíduos doentes apresentam também muitas assimetrias, ao contrário do que acontece com os indivíduos normais. Essas assimetrias traduzem-se, na maioria das componentes obtidas, em claras lateralizações para a esquerda na presença de epilepsia temporal direita e claras lateralizações para a direita na presença de epilepsia temporal esquerda. De notar também que no caso dos doentes, não é possível uma sistematização do processo de redução e escolha de componentes para o *P*3*a* e *P*3*b* devido a estas serem mais difíceis de identificar.

- No caso Normal 2, apesar de ser considerado um caso de um indivíduo normal, as componentes calculadas que compõem o $P300$ apresentam características (topografias da actividade eléctrica e localizações dos geradores) pertencentes a indivíduos doentes. Como já foi referido esta situação verificou-se também em estudos anteriores [23]. Esta situação poderá ser explicada por algum erro no método de ACI utilizado (como por exemplo uma má escolha da distribuição das fontes a separar ou má colocação dos eléctrodos ...), no entanto, nos restantes casos normais estudados neste trabalho, esta situação não se verificou. Poderá ser também explicada por algum erro no registo recolhido, ou então, e esta deverá ser a justificação mais provável, o indivíduo em estudo poderá possuir uma dinâmica cerebral que não é a mesma que a média (em medicina existe uma enorme variabilidade individual, ou seja, cada caso é único e individualizado, pelo que apenas podemos esperar resultados médios (para um conjunto), havendo sempre desvios a essa média). Em último caso, este indivíduo poderá sofrer de alguma anomalia que era desconhecida na altura da recolha dos registos. Questões como esta só poderão ser resolvidas aumentando o número de casos registados, para que haja significância estatística.

- A média do número de componentes presentes nos registos é semelhante tanto no caso dos indivíduos normais como nos doentes. Para achar alguma tendência seria necessário estudar um número bastante superior de casos. Observando os envelopes dos potenciais e o número de componentes presentes para cada um dos casos, verifica-se que no caso do sinal de envelope mostrar muitas variações ou os picos muito prolongados, o número de componentes estimados tende, como seria de esperar, a aumentar.

- O valor do acordo delta (Δ) é sistematicamente menor para o $N100$ relativamente ao $P300$. Esta situação deve-se ao facto de se utilizar o modelo de dipolo duplo para o $N100$ que, para além de tornar o ajuste mais fácil, se adequa mais, à realidade: o $N100$ é composto, de facto, por dois dipolos, enquanto que o $P300$ deverá ser certamente composto por mais do que dois dipolos.

- Nos casos Normal 3 e Normal 4, os resultados obtidos para o potencial $N100$ não foram exactamente os que se esperavam encontrando-se os geradores localizados em áreas demasiado frontais. Esta questão reporta-nos à dificuldade de localizar fontes complexas, tendo apenas acesso a registos com 21 eléctrodos. Se este problema é claramente observado em alguns casos de $N100$, deve fazer-se notar que ele se torna particularmente delicado no que toca ao processamento do $P300$, como se tem feito notar ao longo deste trabalho. O aumento do número de eléctrodos talvez seja a única forma de melhorar esses problemas.

- A localização dos dipolos para o $N100$ mostra não haver grandes variações entre indivíduos normais e doentes. Este potencial foi sempre identificado com dois dipolos (localizados em zonas

temporais medial e frontal) para todos os casos em estudo. Como este potencial aparece sempre e está fundamentalmente relacionado com respostas físicas imediatas, a epilepsia não parece afectar a sua topografia. Ao contrário, dos processos cognitivos subjacentes ao *P*300, que, tal como já se referiu parecem, efectivamente, ser alterados devido à existência de epilepsia.

- Em comparação com estudos anteriores [23], para os casos em estudo, a selecção das componentes relevantes usando um critério de latência foi sempre possível. Como exemplo pode considerar-se os casos dos Doentes 2, 3 e 4 (referentes aos casos 4, 5 e 6 em [23]) onde, neste trabalho, se conseguiu esta selecção das componentes e em [23] não se conseguiu. Esta situação pode dever-se à escolha de um melhor algoritmo de ACI para os casos em estudo e, principalmente, supõe-se que se deva à diminuição do número de componentes, que reduziu tanto os artefactos (como por exemplo potenciais sobrepostos ao potencial de interesse) como o ruído, o que permitiu uma mais fácil identificação.

Outro dos objectivos deste projecto era a construção de uma ferramenta informática com a implementação das técnicas estudadas com a finalidade de facilitar futuros estudos sobre a separação "cega" de sinais, para qualquer tipo de registo (EEG, som, etc). Este objectivo foi atingido através do desenvolvimento do módulo pyIca. Esta ferramenta foi desenvolvida através da implementação dos algoritmos de ACI numa linguagem de programação compilada (C++) com vista a uma execução mais rápida, e através de um *port* das funções criadas para Python para tornar a ferramenta livre e de fácil utilização. Este módulo apresenta várias vantagens em relação aos algoritmos implementados em Matlab. Uma das vantagens é que nos testes efectuados, este módulo provou ser mais rápido que os mesmos algoritmos implementados em Matlab. Outra das vantagens é a utilização de *software* livre e de código aberto no desenvolvimento deste módulo e no desenvolvimento de ferramentas futuras que o utilizem, o que permite uma redução nos custos, mais fácil manutenção e maior facilidade na inclusão de novas funções.

No futuro, e ainda na continuidade deste trabalho, proponho as seguintes melhorias para serem introduzidas:

- Aumentar o número de eléctrodos usados na recolha dos registos para facilitar a localização de geradores neuronais usando o modelo de múltiplos dipolos.

- Aumentar o número de casos estudados para verificar a existência de alguma tendência no número de componentes presentes nos registos de indivíduos normais e doentes. Este aumento no número de casos iria servir, também para:
  - Verificação da existência de perda de componentes de interesse após a redução do número de componentes.
  - Confirmar os resultados referentes às diferenças observadas entre indivíduos normais e doentes.
  - Verificar se o caso anómalo Normal 2 é um caso isolado ou se este se repete para outros indivíduos considerados normais.
- Inclusão de outros algoritmos de ACI no módulo pyIca para aumentar as potencialidades desta ferramenta.

Bibliografia

[1] *Blind source separation and Independent component analysis*. http://www.tsi.enst.fr/~cardoso/guidesepsou.html, acedido em 09-01-2007.

[2] *EEGLAB - Open Source Matlab Toolbox for Electrophysiological Research*. http://www.sccn.ucsd.edu/eeglab/index.html, acedido em 09-01-2007.

[3] *FastICA*. http://www.cis.hut.fi/projects/ica/fastica/, acedido em 09-01-2007.

[4] *ICA:DTU Toolbox*. http://mole.imm.dtu.dk/toolbox/ica/, acedido em 09-01-2007.

[5] *The MathWorks - MATLAB and Simulink for Technical Computing*. http://www.mathworks.com/, acedido em 09-01-2007.

[6] *Numpy Home Page*. `http://numeric.scipy.org`, acedido em 09-01-2007.

[7] *Python Programming Language*. `http://www.python.org`, acedido em 09-01-2007.

[8] *QTriplot*. http://www.ecgsim.org/qtriplot/index.html, acedido em 09-01-2007.

[9] *Robert Oostenveld*. http://oase.uci.ru.nl/~roberto/index.php, acedido em 09-01-2007.

[10] *Simplified Wrapper and Interface Generator*. `http://www.swig.org`, acedido em 09-01-2007.

[11] *Thom Oostendorp*. http://www.mbfys.ru.nl/~thom/, acedido em 09-01-2007.

[12] *Welcome to IT++*. http://itpp.sourceforge.net/latest/, acedido em 09-01-2007.

[13] Guyton A. C. e Hall J. E.: *Textbook of Medical Physiology*. Saunders, 1996.

[14] Jean François Cardoso: *High-order Contrasts for Independent Component Analysis*. Neural Computation, 11:157–192, 1999.

[15] Jean François Cardoso e Antoine Souloumiac: *Blind Beamforming for Non Gaussian Signals*. IEE Proc-F., 40(6):362–370, 1993.

[16] Stok C.J.: *The inverse problem in EEG and MEG with application to visual evoked responses*. Tese de Doutoramento, Universidade de Twente, Holanda, 1986.

[17] Pierre Comon: *Independent component analysis, A new concept?* Signal Processing, 36:287–314, 1994.

[18] Kass R. E. e Raftery A. E.: *Bayes Factors*. Journal of the American Statistical Association, 90(430):773–795, 1995.

[19] Sigurd Enghoff: *Moving ICA and Time-Frequency Analysis in Event-Related EEG Studies of Selective Attention*. Tese de Mestrado, Technical University of Denmark, Department of Physics, 1999.

[20] Mário A. T. Figueiredo, José M. N. Leitão, e Anil K. Jain: *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer-Verlag, páginas 54–69, 1999.

[21] Herrero G., Huupponen E., Värri A., Egiazarian K., Vanrumste B., Vergult A., De Clercq W., Van Huffel S., e Van Paessche: *Independent Component Analysis Of Single Trial Evoked Brain Reponses: Is It Reliable?* In *W. Proceedings of the 2nd International Conference on Computational Intelligence in Medicine and Healthcare*, Costa da Caparica, Lisbon, Portugal, 2005.

[22] Schwarz G.: *Estimating the Dimension of a Model*. Annals of Statistics, 6(2):461–464, 1978.

[23] Rui Alexandre Brandão Arcanjo Galhós: *Utilização da Análise em Componentes Independentes no Processamento de Potenciais Cognitivos P300*. Dissertação de Licenciatura, Universidade do Algarve, 2002.

[24] Jean Gotman: *Current Practice of Clinical Electroencephalography*, capítulo 3 - The Use of Computers in Analysis and Display of EEG and Evoked Potentials. Raven Press, segunda edição, 1990.

[25] Peter D. Grünwald: *Advances in Minimum Description Length Theory and Applications*, capítulo A Tutorial Introduction to the Minimum Description Length Principle. MIT Press, 2005.

[26] Akaike H.: *A New Look at the Statistical Model Identification*. IEEE Transactions on Automatic Control, 19(6):716–723, 1974.

[27] Jasper H. H.: *The ten twenty electrode system of the international federation*. EEG Clinical Neurophysiology, 10:371–375, 1958.

[28] Ulrich Hegerl: *Electroencephalography, Basic Principles, Clinical Applications and Related Fields*, capítulo 31 - Event-Related Potentials in Psychiatry. Lippincott Williams & Wilkins, 1993.

[29] Aapo Hyvärinen e Erkki Oja: *Independent Component Analysis: Algorithms and Applications*. Neural Networks, 13(4-5):411–430, 2000.

[30] Beatty J.: *Principles of Behavorial Neuroscience*. Brown & Benchmark, 1995.

[31] Rissanen J.: *Stochastic Complexity and Modeling*. The Annals of Statistics, 14(3):1080–1100, 1986.

[32] Tortora G. J. e Grabowski S. R.: *Principles of Anatomy and Physiology*. Hasper Collins College Publishers, 1996.

[33] Ray Johnson Jr.: *Progressive Supranuclear Palsy: Clinical and Research Approaches*, capítulo 7 - Event-Related Brain Potentials. New York: Oxford University Press, 1992.

[34] Pratt W. K.: *Digital Image Processing*. John Wiley & Sons, segunda edição, 1991.

[35] Thomas Kolenda, Lars Kai Hansen, e Jan Larsen: *Signal Detection using ICA: Application to Chat Room Topic Spotting*. In *Proceedings of the Third International Conference on Independent Component Analysis and Signal Separation (ICA2001)*, páginas 540–545, San Diego USA, 2001.

[36] Zoltab J. Koles: *Trends in EEG source localization*. Electroencephalography and clinical Neurophysiology, 106:127–137, 1998.

[37] Pizella G. L. e Raftery A. E.: *Principles of Magnetoencephalography*. Advances in Neurology, 54, 1990.

[38] Scott Makeig, Marissa Westerfield, Tzyy Ping Jung, James Covington, Jeanne Townsend, Terrence J. Sejnowski, e Eric Courchesne: *Functionally Independent Components of the Late Positive Event-Related Potential during Visual Spatial Attention*. The Journal of Neuroscience, 19(7):2665–2680, 1999.

[39] Ernst Niedermeyer: *Electroencephalography, Basic Principles, Clinical Applications and Related Fiels*, capítulo 28 - Epilepctic Seizure Disorders. Lippincott Williams & Wilkins, 1993.

[40] Ernst Niedermeyer: *Electroencephalography, Basic Principles, Clinical Applications and Related Fiels*, capítulo 14 - Abnormal EEG Patterns: Epileptic and Paroxysmal. Lippincott Williams & Wilkins, 1993.

[41] Hans Bruun Nielsen: *UCMINF - An Algorithm for Unconstrained, Nonlinear Optimization*. Relatório Técnico 200019, IMM Department of Mathematical Modelling, Technical University of Denmark, 2001.

[42] Paul L. Nunez: *Electric Fields of The Brain*. Oxford University Press, 1981.

[43] J. J. Oliver, R. A. Baxter, e C. S. Wallace: *Unsupervised Learning Using MML.* In *Machine Learning: Proceedings of the Thirteenth International Conference (ICML 96)*, páginas 364–372. Morgan Kaufmann Publishers, 1996.

[44] Laarne P.H., Tenhunen Eskelinen M.L., Hyttinen J.K., e Eskola H.J.: *Effect of EEG Electrode Density on Dipole Localization Accuracy Using Two Realistically Shaped Skull Resistivity Models*. Springer: Brain Topography, 12(4):249–254(6), 2000.

[45] Kandel E. R. e Schwartz J. H.: *Principles of Neural Science*. Elsevier, 1985.

[46] Edward L. Reilly: *Electroencephalography, Basic Principles, Clinical Applications and Related Fields*, capítulo 7 - EEG Recordings and Operation of the Apparatus. Lippincott Williams & Wilkins, 1993.

[47] Michael Scherg, Jiri Vajsar, e Terence W. Picton: *A Source Analisys of the Late Human Auditory Evoked Potencials*. Journal of Cognitive Neuroscience, 1(4):336–355, 1989.

[48] Rob R. Seeley, Trent D. Stephens, e Philip Tate: *Anatomia e Fisiologia*. Mosby, terceira edição, 1995.

[49] C. Silva, R. Almeida, T. Ooostendorp, E. Ducla-Soares, J. P. Foreid, e T. Pimentel: *Interictal spike localization using a standard realistic head model: simulations and analysis of clinical data*. Clinical Neurophysiology, 1110:846–855, 1999.

[50] Carla Silva: *Processamento de Dados Electroencefalográficos - aplicações à epilepsia*. Tese de Doutoramento, Universidade de Lisboa, 1998.

[51] Fernando Lopes da Silva: *Electroencephalography, Basic Principles, Clinical Applications and Related Fields*, capítulo 50 - Event-Related Poptentials: Methodology and Quantification. Lippincott Williams & Wilkins, 1993.

[52] Mark Girolami Te-Won Lee e Terrence J. Sejnowski: *Independent Component Analysis Using an Extended Infomax Algorithm for Mixed Subgaussian and Supergaussian Sources*. Neural Computation, 11:417–441, 1999.

[53] Arthur Vander, James Sherman, e Dorothy Luciano: *Human Physiology The Mechanisms of Body Function*. WCB/McGraw-Hill, sétima edição, 1998.

[54] F. Vrins, J. A. Lee, M. Verleysen, V. Vigneron, e C. Jutten: *Improving Independent Component Analysis Performances By Variable Selection*. In *NNSP'2003 Proceedings, Neural Networks for Signal Processing, IEEE*, páginas 359–368.

[55] Vários: *Le Cerveau*. Bibliothèque pour la Science, 1984.

Anexo A: FastIca

Para o cálculo da matriz de separação W, o algoritmo FastIca utiliza para a medida da não-Gaussianidade uma aproximação à negentropia (entropia negativa), que neste caso é uma aproximação baseada no principio da máxima entropia, que é dada pela expressão A.1 [29].

$$J(y_i) \propto (E\{G(y_i)\} - E\{G(\nu)\})^2, \tag{A.1}$$

em que $\propto$ representa proporcionalidade, E a esperança matemática, y é uma componente que queremos calcular, que é igual a $y_i = w_i^T X$ e ν é uma variável gaussiana de média zero e variância unitária fazendo assim o termo $E\{G(\nu)\}$ constante. $G(y_i)$ é uma função não-quadrática cuja escolha depende do problema que se pretende resolver. Algumas das funções mais utilizadas são [29]:

$$G_1(y_i) = \frac{1}{a_1} log(cosh(a_1 y_i)), \tag{A.2}$$

$$G_2(y_i) = -e^{\frac{1}{2}y_i^2}, \tag{A.3}$$

$$G_3(y_i) = y_i^4, \tag{A.4}$$

A função $J(y_i)$ funcionará como medida de Gaussianidade, sendo 0 para componentes gaussianas. Quanto maior for o seu valor, maior será a não-Gaussianidade de y_i [29].

O FastIca é um algoritmo baseado num esquema de iteração de ponto-fixo para achar a máxima não-Gaussianidade de $w_i^T x$ através da medição A.1. Este algoritmo pode ser derivado da seguinte forma: estabelecendo para uma única componente que w é um vector de W, ou seja, $w = w_i$, o máximo da negentropia de $w^T X$ é obtido num certo valor óptimo de $E\{G(w^T X)\}$. De acordo com as condições de Kuhn-Tucker, os valores óptimos de $E\{G(w^T X)\}$ com a condição de $E\{(w^T X)^2)\} = \|w\|^2 = 1$ são obtidos quando se verifica:

$$E\{xg(w^T X)\} - \beta w = 0, \tag{A.5}$$

em que $g(y)$ é a primeira derivada da função não-quadrática $G(y)$. Resolvendo a equação A.5 através do método de Newton e denotando o lado esquerdo da equação A.5 como F, obtemos a seguinte matriz Jacobiana [29]:

$$JF(w) = E\{XX^T g'(w^T X)\} - \beta I = 0, \tag{A.6}$$

Para simplificar a inversão desta matriz, faz-se uma aproximação do primeiro termo de A.6 [29]. Como os dados foram previamente branqueados, uma aproximação razoável que pode ser feita é considerar que:

$$E\{XX^T g'(w^T X)\} \approx E\{XX^T\}E\{g'(w^T X)\} = E\{g'(w^T X)\}I. \tag{A.7}$$

Com esta aproximação a matriz Jacobiana torna-se diagonal o que facilita a sua inversão. Assim obtém-se a seguinte aproximação da iteração de Newton [29]:

$$w^+ = w - \frac{E\{Xg'(w^T X)\} - \beta w}{E\{g'(w^T X) - \beta\}}, \tag{A.8}$$

em que w^+ denota a iteração seguinte de w. Este algoritmo pode ainda ser mais simplificado através da multiplicação dos dois lados da equação A.8 por $(\beta - E\{g'(w^T X))$ [29]. Depois de uma simplificação algébrica obtém-se a seguinte expressão:

$$w^+ = E\{Xg(w^T X)\} - E\{g'(w^T X)\}w, \tag{A.9}$$

chamada de iteração do FastIca [29]. Na prática as esperanças matemáticas de A.9 são substituídas por aproximações, sendo neste caso médias.

Resumindo, o algoritmo FastIca para uma só componente segue os seguintes passos:

1. Escolher vector inicial w (pode ser aleatório).
2. $w^+ = E\{Xg(w^T X)\} - E\{g'(w^T X)\}w$
3. $w = \frac{w^+}{\|w^+\|}$
4. Se não convergir, voltar para o ponto 2.

De notar que convergência aqui significa que os novos e os antigos valores de w apontam para a mesma direcção, ou seja, o seu produto interno é igual ou quase igual a 1 [29].

Mas normalmente queremos calcular mais do que uma componente, e, portanto, é necessário prevenir que diferentes vectores convirjam para o mesmo máximo, para isso é necessário *descorrelacionar* as saídas $w_1^T X, \ldots, w_n^T X$ depois de cada iteração [29]. Existem pelo menos dois métodos para atingir

esse objectivo. No primeiro, a descorrelação é feita através de um esquema de deflacção baseado na descorrelação de Gram-Schmidt [29]. A utilização deste método implica que as componentes independentes sejam calculadas uma a uma. Quando já foram estimadas p componentes ou p vectores $w_1, \ldots, w_p$, corre-se o algoritmo anterior para w_{p+1}, e depois de cada iteração no cálculo de w_{p+1} devem ser subtraídas as projecções $w_{p+1}^T w_j w_j$, $j = 1, \ldots, p$ dos p vectores anteriormente calculados, e depois re-normaliza-se w_{p+1}:

1. $w_{p+1} = w_{p+1} - \sum_{j=1}^{p} w_{p+1}^T w_j w_j$
2. $w_{p+1} = \frac{w_{p+1}}{\sqrt{w_{p+1}^T w_{p+1}}}$.

No segundo método de descorrelação, a descorrelação é simétrica, onde nenhum vector é privilegiado sobre os outros. Neste caso, os vectores w_i vão ser todos calculados ao mesmo tempo, ou seja, os elementos da matriz de separação W são obtidos em simultâneo. Para fazer este cálculo basta seguir o seguinte algoritmo:

1. $W = \frac{W}{\sqrt{\|WW^T\|}}$
2. $W = \frac{3}{2}W - \frac{1}{2}WW^TW$
3. Repetir o ponto 2 atá convergir.

Em qualquer dos métodos, no final, a matriz de separação calculada deve ser multiplicada pela matriz de branqueamento, ou seja, $W\,W_h$.

Anexo B: IcaML (Infomax)

O algoritmo IcaML utiliza a maximização da log-verosimilhança para a estimação das fontes. Como é explicado em [29] e em [17], a maximização da log-verosimilhança é equivalente à minimização da informação mútua. Considerando que a densidade conjunta das misturas é dada por [52]:

$$p(x) = |det(W)|p(y), \tag{B.1}$$

sendo $p(y) = \prod_{i=1}^{N} p_i(y_i)$ a hipotética distribuição conjunta de $p(s)$, a log-verosimilhança é dada pela equação B.2 [52].

$$L(y, W) = \log|det(W)| + \sum_{i=1}^{N} \log p_i(y_i). \tag{B.2}$$

A partir da maximização de B.2 em ordem a W, obtém-se a seguinte regra de actualização [52]:

$$\Delta W \propto \left[(W^{-1})^T - \varphi(y)x^T\right], \tag{B.3}$$

onde [52]:

$$\varphi(y) = \frac{-\frac{\partial p(y)}{\partial y}}{p(y)}\left[\frac{-\frac{\partial p(y_1)}{\partial y_1}}{p(y_1)}, \frac{-\frac{\partial p(y_2)}{\partial y_2}}{p(y_2)}, \ldots, \frac{-\frac{\partial p(y_N)}{\partial y_N}}{p(y_N)}\right]^T, \tag{B.4}$$

no caso de ser utilizada uma $p(y)$ prévia, que será uma hipotética distribuição de $p(s)$. O algoritmo IcaML faz uso da seguinte distribuição:

$$p(y) = \prod_{i=1}^{N} \frac{1}{\pi \cosh y_i}. \tag{B.5}$$

Com o uso desta densidade para as componentes estimadas, apenas as fontes com distribuição super--Gaussiana vão ser recuperadas.

Existem várias formas de maximização da log-verosimilhança com respeito a W mais eficientes. No algoritmo IcaML, a maximização da log-verosimilhança é feita de uma forma mais precisa através da utilização do algoritmo UCMINF (algoritmo tipo BFGS (Broyden-Fletcher-Goldfarb-Shanno)) [35] que utiliza optimizações não-lineares para o cálculo de mínimos locais [41].

No algoritmo IcaML, devido ao facto de este utilizar o UCMINF para atingir a convergência, para evitar problemas numéricos, os sinais misturados são normalizados através da divisão de todos os valores dos sinais pelo máximo valor absoluto dos dados [52].

Anexo C: Jade

Para o cálculo da matriz de separação W, o algoritmo Jade apresenta uma ligeira diferença em relação aos outros algoritmos de ACI. Enquanto os outros algoritmos optimizam uma transformada particular dos dados observados (entrada), o Jade optimiza uma transformada de um conjunto particular de estatísticas dos dados de entrada. O ponto de partida do Jade é o facto que a generalidade dos algoritmos de ACI necessitarem de uma estimação das distribuições das componentes independentes (fontes) ou assumirem um determinado tipo de distribuição das fontes. O Jade consegue fazer a separação das componentes através da optimização das aproximações dos cumulantes dos dados, não precisando, desta forma, de qualquer tipo de informação sobre a distribuição das fontes. O Jade é um algoritmo que utiliza, como já foi dito, medidas teóricas de informação que operam sobre os cumulantes cruzados de quarta ordem dos dados [14].

Todas as medidas teóricas de informação podem ser calculadas através de operações com os cumulantes [14]. Uma das medidas de não-Gaussianidade, a Kurtosis, é os auto-cumulantes de quarta ordem. A matriz de cumulantes com elementos $[Q^x(M)]_{ij}$ é definida como [14]:

$$[Q^x(M)]_{ij} = \sum_{k=l=1}^{n} Cum(x_i, x_j, x_k, x_l) M_{kl}, \tag{C.1}$$

onde n é o número de componentes, M é uma matriz $n \times n$ e x é um vector (coluna da matriz X) $n \times 1$. As matrizes de cumulantes são calculadas a partir dos dados branqueados (com ou sem redução) X. O algoritmo Jade calcula a matriz de separação W através da multiplicação de $\hat{V}^T W_h$, onde V é uma matriz de rotação que faz com que a matriz de cumulantes cruzados seja o mais diagonal possível [14] e é definida por:

$$\hat{V} = argmin \sum_{i} Off(V^T Q^X(M_i) V), \tag{C.2}$$

onde $Off(F) = \sum_{i \neq j} (f_{ij})^2$ e serve de medida da não-diagonalidade da matriz F. Resumindo, esta não-diagonalização não é mais do que uma optimização de uma função de contraste que neste caso é denotada como $\phi^{JADE}(Y)$ e é definida pela equação C.3 [14]:

$$\phi^{JADE}(Y) = Off(Q^Y(M)) = \sum_{ijkl \neq iikl} (Q^Y(M_{ijkl}))^2. \tag{C.3}$$

A função $\phi^{JADE}(Y)$ serve de medida da informação mútua das componentes independentes $Y = WX$. Assim, quanto menor for o valor $\phi^{JADE}(Y)$, maior é a não-diagonalização das matrizes de cumulantes cruzados das componentes Y e menor é a informação mútua entre as componentes Y [14].

Anexo D: Bayesian Information Criterion

Considerando um conjunto de modelos $m = 0, \ldots, M$, pretende-se maximizar a probabilidade de um modelo m, para um conjunto de dados observados X. Usando a relação de Bayes, esta probabilidade pode ser escrita através da expressão 5.1 [35].

$$p(m|X) = \frac{p(X|m)p(m)}{p(X)}, \tag{D.1}$$

onde $p(m)$ é a função densidade de probabilidade do modelo.

Um modelo é normalmente definido por um conjunto de parâmetros θ. Sabendo isto, fica-se com a densidade de probabilidade $p(X|\theta, m)$. A relação D.2 é então obtida [35]:

$$p(X|m) = \int p(X, \theta|m)d\theta = \int p(X|\theta, m)p(\theta|m)d\theta. \tag{D.2}$$

A distribuição $p(\theta|m)$ representa a densidade de probabilidade dos parâmetros para um determinado modelo e muitas vezes é assumido que não tem grande influência no integral D.2. O integral da equação D.2 é, por vezes, muito complicado para ser resolvido de uma forma analítica [35]. Foram sugeridos vários esquemas de aproximação a este integral. O método BIC (Bayesian Information Criterion) faz a seguinte aproximação ao integral [22][35]:

$$p(X|m) \approx p(X|\theta, m)p(\theta, m)(N)^{-\frac{d}{2}}, \tag{D.3}$$

em que d é a dimensão do vector de parâmetros θ, e N é o número de amostras de cada sinal. No caso do modelo da ACI, $p(\theta, m) = 1$ devido ao facto de todos os parâmetros serem conhecidos [35].

O critério BIC é baseado no indicador *bic* dado pela expressão D.4 [22]:

$$bic = -log(p(X|\theta, m)) + \frac{d}{2}log(N). \tag{D.4}$$

Denotando agora o modelo pelo número de componentes que se pretende (sub-espaços), temos $m = K$ onde K é o número de componentes a separar, $p(X|\theta, K)$ é definido pela expressão D.5[35]:

$$p(X|\theta, K) = p(X|W, K)p(\varepsilon|\sigma_\varepsilon^2), \tag{D.5}$$

onde $p(X|\theta, K)$ é a densidade de probabilidade da ACI para K componentes. Como foi aplicada a ACP para reduzir os dados através da escolha dos valores próprios mais elevados da matriz de covariância de X, os restantes valores próprios são considerados como representantes dos sub-espaços de ruído (ε). A densidade de probabilidade do ruído é então dada por [35]:

$$p(\varepsilon|\sigma_\varepsilon^2) = (2\pi\sigma_\varepsilon^2)^{-\frac{N(M-K)}{2}} . \exp(\frac{-N(M-K)}{2}), \tag{D.6}$$

onde $\sigma_\varepsilon^2 = \frac{1}{N(M-K)} \sum_{i=K+1}^{N} D_i^2$ e M é o número de misturas. Como $\theta = (W_h, \sigma_\varepsilon^2, W)$, o número total de parâmetros é $d = K(2M - K + 1)/2 + 1 + MK$.

Anexo E: Definição de Cumulantes

Considerando x como sendo uma variável escalar aleatória de média zero cuja função característica é denotada por $\hat{f}$:

$$\hat{f}(\mu) = E\{e^{j\mu x}\}. \tag{E.1}$$

Se for feita a expansão do logaritmo desta função característica como uma série de Taylor, obtém-se:

$$log(\hat{f}(\mu)) = k_1(j\mu) + k_2\frac{(j\mu)^2}{2} + \ldots + k_n\frac{(j\mu)^n}{n!} + \ldots, \tag{E.2}$$

onde k_n são constantes. Estas constantes são chamadas de cumulantes da distribuição de x [14]. Os primeiros três cumulantes (para variáveis de média zero) têm as seguintes expressões:

$$k_1 = E\{x\} = 0 \tag{E.3}$$

$$k_2 = E\{x^2\} \tag{E.4}$$

$$k_3 = E\{x^3\}. \tag{E.5}$$

O cumulante de quarta ordem é chamado de *Kurtosis* e pode ser considerado como medida de não--Gaussianidade. O cumulante de quarta ordem é dado pela expressão E.6 [14]:

$$k_4 = kurt(x) = E\{x^4\} - 3(E\{x^2\})^2. \tag{E.6}$$

Anexo F: Documentação da biblioteca icapp

F.1 Referência ao ficheiro icapp.h

Namespaces

- namespace **std**

Componentes

- class **Mix**

  Classe que guarda os sinais misturados.

- class **Ica**

  Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

- class **Fastica**

  Sub-classe da classe **Ica**(p. 126) *que implementa o algoritmo Fastica.*

- class **IcaML**

  Sub-classe da classe **Ica**(p. 126) *que implementa o algoritmo IcaML.*

F.1.1 Descrição detalhada

Autor:

Michael Guerreiro

Data:

08/02/2006

F.2 Hierarquia de classes

Esta lista de heranças está organizada, dentro do possível, por ordem alfabética:

F.3 Lista de componentes

Lista de classes, estruturas, uniões e interfaces com uma breve descrição:

F.4 Referência à classe Mix

Classe que guarda os sinais misturados.

`#include <icapp.h>`

Membros públicos

- **Mix** ()

  Constructor base.

- **Mix** (const **Mix** &other)

  Construtor cópia com um parâmetro que cria um objecto cópia de outro objecto do tipo Mix.

- **Mix** (const mat &mtx)

  Construtor com um parâmetro que cria um objecto utilizando os dados de uma matriz do tipo mat (it++).

- **Mix** (char ∗lfilename, const int mixt, const int samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- virtual ~**Mix** ()

  Destrutor.

- virtual void **load_matrix** (char ∗lfilename, const int mixt, const int samples)

  Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

- virtual void **save_matrix** (char ∗lfilename)

  Função com três parâmetros que grava os dados das misturas do objecto para um ficheiro.

- virtual int **get_samples** (void) const

  Função que retorna o número de instantes.

- virtual int **get_mixt** (void) const

Função que retorna o número de misturas.

- virtual vec **get_row** (int i) const

  Função com um parâmetro que retorna um vector contendo uma das misturas.

- virtual vec **get_col** (int i) const

  Função com um parâmetro que retorna um vector contento o valor de todas as misturas para um determinado instante.

- virtual mat **get_data** (void) const

  Função que retorna uma matriz contendo os dados das misturas.

- virtual char * **get_filename** (void) const

  Função que retorna o nome do ficheiro de onde foram retirados os dados das misturas.

Atributos Públicos

- mat **matrix**

  Variável do tipo mat (it++) contendo a matriz das misturas..

Membros protegidos

- virtual void **reset** (void)

  Função que coloca os atributos da classe aos seus valores por defeito.

- virtual void **set_matrix** (const mat &mtx)

  Função com um parâmetro que atribui os dados das misturas à classe.

Atributos Protegidos

- int **m**

Variável do tipo int contendo o número de misturas.

- int **s**

  Variável do tipo int contento o número de amostras (instantes).

- char * **filename**

  Ponteiro do tipo char apontando para o nome do ficheiro onde estão registadas as misturas.

F.4.1 Descrição detalhada

Classe que guarda os sinais misturados.

É utilizada como entrada nas funções da classe **Ica**(p. 126) e suas sub-classes.

F.4.2 Documentação dos Construtores & Destrutor

F.4.2.1 Mix::Mix ()

Constructor base.

F.4.2.2 Mix::Mix (const Mix & *other*)

Construtor cópia com um parâmetro que cria um objecto cópia de outro objecto do tipo Mix.

Parâmetros:

other Endereço do tipo Mix de um objecto da classe Mix.

F.4.2.3 Mix::Mix (const mat & *mtx*)

Construtor com um parâmetro que cria um objecto utilizando os dados de uma matriz do tipo mat (it++).

Parâmetros:

mtx Uma variável do tipo mat (it++) contendo os dados das misturas.

F.4.2.4 Mix::Mix (char * *lfilename*, const int *mixt*, const int *samples*)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

mixt Uma variável do tipo int contendo o número de misturas.

samples Uma variável do tipo int contento o número de amostras (instantes).

F.4.2.5 virtual Mix::~Mix () [virtual]

Destrutor.

F.4.3 Documentação dos métodos

F.4.3.1 virtual void Mix::load_matrix (char * *lfilename*, const int *mixt*, const int *samples*) [virtual]

Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

mixt Uma variável do tipo int contendo o número de misturas.

samples Uma variável do tipo int contento o número de amostras (instantes).

F.4.3.2 virtual void Mix::save_matrix (char * *lfilename*) [virtual]

Função com três parâmetros que grava os dados das misturas do objecto para um ficheiro.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

F.4.3.3 virtual int Mix::get_samples (void) const [virtual]

Função que retorna o número de instantes.

Retorna:

Uma variável do tipo int contendo o número de instantes.

F.4.3.4 virtual int Mix::get_mixt (void) const [virtual]

Função que retorna o número de misturas.

Retorna:

Uma variável do tipo int contendo o número de misturas.

F.4.3.5 virtual vec Mix::get_row (int *i*) const [virtual]

Função com um parâmetro que retorna um vector contendo uma das misturas.

Parâmetros:

i Uma variável do tipo int contendo o número da mistura.

Retorna:

Uma variável do tipo vec (it++) contendo uma mistura.

F.4.3.6 virtual vec Mix::get_col (int *i*) const [virtual]

Função com um parâmetro que retorna um vector contento o valor de todas as misturas para um determinado instante.

Parâmetros:

i Uma variável do tipo int contendo o número do instante.

Retorna:

Uma variável do tipo vec (it++) contendo o valor de todas as misturas para o instante i.

F.4.3.7 virtual mat Mix::get_data (void) const [virtual]

Função que retorna uma matriz contendo os dados das misturas.

Retorna:

Uma variável do tipo mat (it++) contendo os dados das misturas.

F.4.3.8 virtual char* Mix::get_filename (void) const [virtual]

Função que retorna o nome do ficheiro de onde foram retirados os dados das misturas.

Retorna:

Um ponteiro do tipo char apontando para o nome do ficheiro.

F.4.3.9 virtual void Mix::reset (void) [protected, virtual]

Função que coloca os atributos da classe aos seus valores por defeito.

F.4.3.10 virtual void Mix::set_matrix (const mat & *mtx*) [protected, virtual]

Função com um parâmetro que atribui os dados das misturas à classe.

Parâmetros:

mtx Um endereço do tipo mat (it++) de um objecto da classe mat (it++) contendo os dados das misturas.

F.4.4 Documentação dos dados membro

F.4.4.1 int Mix::m [protected]

Variável do tipo int contendo o número de misturas.

F.4.4.2 int Mix::s [protected]

Variável do tipo int contento o número de amostras (instantes).

F.4.4.3 char* Mix::filename [protected]

Ponteiro do tipo char apontando para o nome do ficheiro onde estão registadas as misturas.

F.4.4.4 mat Mix::matrix

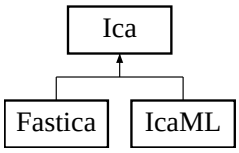
Variável do tipo mat (it++) contendo a matriz das misturas.

F.5 Referência à classe Ica

Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

`#include <icapp.h>`

Diagrama de heranças da classe Ica:



Membros públicos

- **Ica** ()

  Constructor base.

- virtual ~**Ica** ()

  Destrutor.

- virtual void **load_matrix** (char ∗lfilename, const int mixt, const int samples)

  Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

- virtual void **set_ncomp** (int ncomp)

  Função com um parâmetro que modifica o número de componentes a calcular.

- virtual int **get_ncomp** (void) const

  Função que retorna o número de componentes.

- virtual void **calc_comp** (void)

  Função que calcula as componentes a partir das misturas e da matriz de separação.

- virtual void **calc_icaproj** (const int comp)

  Função com um parâmetro que calcula as projecções de uma determinada componente.

- virtual mat **get_sepmat** (void)

  Função que retorna uma matriz contendo a matriz de separação.

- virtual mat **get_mixtmat** (void)

  Função que retorna uma matriz contendo a matriz misturas.

- virtual mat **get_comp** (void)

  Função que retorna uma matriz contendo as componentes calculadas.

- virtual mat **get_proj** (void)

  Função que retorna uma matriz contendo as projecções de uma determinada componente.

- virtual mat **get_data** (void)

  Função que retorna uma matriz contendo os dados das misturas.

Membros protegidos

- virtual void **reset** (void)

  Função que coloca os atributos da classe aos seus valores por defeito.

- virtual void **rmmean** (mat &Xm, const mat &Xv)

  Função com dois parâmetros que remove a média das misturas.

- virtual double **scale_x** (mat &Xs, const mat &Xm)

  Função com dois parâmetros que faz o escalonamento dos dados de mistura removendo o valor máximo.

- virtual void **pca** (mat &Wh, const mat &Xm, const int &mixt, const int &samples)

  Função com dois parâmetros que faz o calculo da ACP com ou sem redução de componentes.

Atributos Protegidos

- int **n**

  Variável do tipo int contendo o número de componentes a calcular.

- **Mix** ∗ **X**

  Variável do tipo **Mix**(p. 120) *contendo as misturas.*

- mat ∗ **W**

  Variável do tipo mat (it++) contendo a matriz de separação calculada.

- mat ∗ **A**

  Variável do tipo mat (it++) contendo a matriz de mistura calculada.

- mat ∗ **Y**

  Variável do tipo mat (it++) contendo as componentes calculadas.

- mat ∗ **ICAproj**

  Variável do tipo mat (it++) contendo as projecções de uma determindada componente.

F.5.1 Descrição detalhada

Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

Esta classe serve de base para as classes que implementam os algortimos de ACI.

F.5.2 Documentação dos Construtores & Destrutor

F.5.2.1 Ica::Ica ()

Constructor base.

F.5.2.2 virtual Ica::~Ica () [virtual]

Destrutor.

F.5.3 Documentação dos métodos

F.5.3.1 virtual void Ica::load_matrix (char * *lfilename*, const int *mixt*, const int *samples*) [virtual]

Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

mixt Uma variável do tipo int contendo o número de misturas.

samples Uma variável do tipo int contento o número de amostras (instantes).

F.5.3.2 virtual void Ica::set_ncomp (int *ncomp*) [virtual]

Função com um parâmetro que modifica o número de componentes a calcular.

Parâmetros:

ncomp Uma variável do tipo int contendo o número de componentes.

F.5.3.3 virtual int Ica::get_ncomp (void) const [virtual]

Função que retorna o número de componentes.

Retorna:

Uma variável do tipo int contendo o número de componentes.

F.5.3.4 virtual void Ica::calc_comp (void) [virtual]

Função que calcula as componentes a partir das misturas e da matriz de separação.

F.5.3.5 virtual void Ica::calc_icaproj (const int *comp*) [virtual]

Função com um parâmetro que calcula as projecções de uma determinada componente.

Parâmetros:

comp Uma variável do tipo int contendo o número da componente.

F.5.3.6 virtual mat Ica::get_sepmat (void) [virtual]

Função que retorna uma matriz contendo a matriz de separação.

Retorna:

Uma variável do tipo mat (it++) contendo a matriz de separação.

F.5.3.7 virtual mat Ica::get_mixtmat (void) [virtual]

Função que retorna uma matriz contendo a matriz misturas.

Retorna:

Uma variável do tipo mat (it++) contendo a matriz de misturas.

F.5.3.8 virtual mat Ica::get_comp (void) [virtual]

Função que retorna uma matriz contendo as componentes calculadas.

Retorna:

Uma variável do tipo mat (it++) contendo as componentes calculadas.

F.5.3.9 virtual mat Ica::get_proj (void) [virtual]

Função que retorna uma matriz contendo as projecções de uma determinada componente.

Retorna:

Uma variável do tipo mat (it++) contendo as projecções de uma determinada componente.

F.5.3.10 virtual mat Ica::get_data (void) [virtual]

Função que retorna uma matriz contendo os dados das misturas.

Retorna:

Uma variável do tipo mat (it++) contendo os dados das misturas.

F.5.3.11 virtual void Ica::reset (void) [protected, virtual]

Função que coloca os atributos da classe aos seus valores por defeito.

F.5.3.12 virtual void Ica::rmmean (mat & *Xm*, const mat & *Xv*) [protected, virtual]

Função com dois parâmetros que remove a média das misturas.

Parâmetros:

Xm Um endereço do tipo mat (it++).

Xv Um endereço do tipo mat (it++) contendo os dados das misturas.

Retorna:

O resultado é retornado no endereço de Xm.

F.5.3.13 virtual double Ica::scale_x (mat & *Xs*, const mat & *Xm*) [protected, virtual]

Função com dois parâmetros que faz o escalonamento dos dados de mistura removendo o valor máximo.

Parâmetros:

Xs Um endereço do tipo mat (it++).

Xm Um endereço do tipo mat (it++) contendo os dados das misturas.

Retorna:

O resultado é retornado no endereço de Xs.

F.5.3.14 virtual void Ica::pca (mat & *Wh*, const mat & *Xm*, const int & *mixt*, const int & *samples*) [protected, virtual]

Função com dois parâmetros que faz o calculo da ACP com ou sem redução de componentes.

Parâmetros:

Wh Um endereço do tipo mat (it++).

Xm Um endereço do tipo mat (it++) contendo os dados das misturas.

mixt Um endereço do tipo int contendo o número de misturas.

samples Um endereço do tipo int contendo o número de instantes.

Retorna:

A matriz de branqueamento é retornada Xm.

F.5.4 Documentação dos dados membro

F.5.4.1 int Ica::n [protected]

Variável do tipo int contendo o número de componentes a calcular.

F.5.4.2 Mix* Ica::X [protected]

Variável do tipo **Mix**(p. 120) contendo as misturas.

F.5.4.3 mat* Ica::W [protected]

Variável do tipo mat (it++) contendo a matriz de separação calculada.

F.5.4.4 mat* Ica::A [protected]

Variável do tipo mat (it++) contendo a matriz de mistura calculada.

F.5.4.5 mat* Ica::Y [protected]

Variável do tipo mat (it++) contendo as componentes calculadas.

F.5.4.6 mat* Ica::ICAproj [protected]

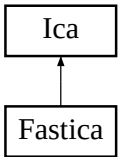
Variável do tipo mat (it++) contendo as projecções de uma determindada componente

F.6 Referência à classe Fastica

Sub-classe da classe **Ica**(p. 126) que implementa o algoritmo Fastica.

`#include <icapp.h>`

Diagrama de heranças da classe Fastica:



Membros públicos

- **Fastica** (const **Mix** &x)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 120).

- **Fastica** (char ∗lfilename, const int mixt, const int samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- virtual ~**Fastica** ()

  Destrutor.

- virtual void **separate** (void)

  Função que executa o algoritmo de ACI e calcula a matriz de separação.

- virtual void **set_maxiterations** (const int it)

  Função com um paràmetro que modifica o número de iterações do algoritmo.

- virtual void **set_nonlinearity** (const int ig)

  Função com um parâmetro que modifica a não-linearidade que o algoritmo vai utilizar.

- virtual void **set_approach** (const int method)

  Função com um parâmetro que modifica o tipo de aproximação do algoritmo.

- virtual void **set_initguess** (char *matrix)

  Função com um parâmetro que modifica a estimativa da matriz de separação inicial.

- virtual void **reset_initguess** ()

  Função que coloca a matriz de separação inicial aos valores por defeito.

- virtual void **set_a1** (const double la1)

  Função com um parâmetro que modifica o valor da variável a1 na não-linearidade Tanh.

- virtual void **set_a2** (const double la2)

  Função com um parâmetro que modifica o valor da variável a2 na não-linearidade Gauss.

- virtual int **get_nonlinearity** (void) const

  Função que retorna a não-linearidade que algoritmo está a utilizar.

- virtual int **get_approach** (void) const

  Função que retorna a aproximação que o algoritmo esta a utilizar.

- virtual int **get_maxiterations** (void) const

  Função que retorna o número de iterações a que está regulado o algoritmo.

- virtual double **get_a1** (void) const

  Função que retorna o valor de a1 da não-linearidade Tanh.

- virtual double **get_a2** (void) const

  Função que retorna o valor de a2 da não-linearidade Gauss.

Membros privados

- virtual void **reset_fastica** (void)

  Função que coloca os atributos da classe aos seus valores por defeito.

- virtual void **init_fastica** (void)

  Função que faz a inicialização dos atributos da classe.

Atributos Privados

- int **maxit**

  Variável do tipo int contendo o número máximo de iterações.

- int **g**

  Variável do tipo int contendo a não-lineridade (10 - Pow3; 20 - Tanh; 30 - Gauss).

- int **approach**

  Variável do tipo int contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

- mat **initguess**

  Variável do tipo mat (it++) contendo a primeira estimativa da matriz de separação.

- double **epsilon**

  Variável do tipo double contendo o menor valor considerado.

- int **a1**

  Variável do tipo int contendo o valor da variável a1 na não-linearidade Tanh.

- int **a2**

  Variável do tipo int contendo o valor da variável a2 na não-linearidade Gauss.

F.6.1 Descrição detalhada

Sub-classe da classe **Ica**(p. 126) que implementa o algoritmo Fastica.

F.6.2 Documentação dos Construtores & Destrutor

F.6.2.1 Fastica::Fastica (const Mix & *x*)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 120).

Parâmetros:

x Um objecto da classe **Mix**(p. 120) contendo as misturas..

F.6.2.2 Fastica::Fastica (char * *lfilename*, const int *mixt*, const int *samples*)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

mixt Uma variável do tipo int contendo o número de misturas.

samples Uma variável do tipo int contento o número de amostras (instantes).

F.6.2.3 virtual Fastica::~Fastica () `[virtual]`

Destrutor.

F.6.3 Documentação dos métodos

F.6.3.1 virtual void Fastica::separate (void) `[virtual]`

Função que executa o algoritmo de ACI e calcula a matriz de separação.

F.6.3.2 virtual void Fastica::set_maxiterations (const int *it*) `[virtual]`

Função com um parâmetro que modifica o número de iterações do algoritmo.

Parâmetros:

it Uma variável do tipo int contendo o número de iterações.

F.6.3.3 virtual void Fastica::set_nonlinearity (const int *ig*) [virtual]

Função com um parâmetro que modifica a não-linearidade que o algoritmo vai utilizar.

Parâmetros:

ig Uma variável do tipo int contendo o tipo de não-linearidade (10 - Pow3; 20 - Tanh; 30 - Gauss).

F.6.3.4 virtual void Fastica::set_approach (const int *method*) [virtual]

Função com um parâmetro que modifica o tipo de aproximação do algoritmo.

Parâmetros:

method Uma variável do tipo int contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

F.6.3.5 virtual void Fastica::set_initguess (char * *matrix*) [virtual]

Função com um parâmetro que modifica a estimativa da matriz de separação inicial.

Parâmetros:

matrix Uma ponteiro do tipo mat (it++) que aponta para uma variável contendo uma estimativa da matriz de separação inicial.

F.6.3.6 virtual void Fastica::reset_initguess () [virtual]

Função que coloca a matriz de separação inicial aos valores por defeito.

F.6.3.7 virtual void Fastica::set_a1 (const double *la1*) [virtual]

Função com um parâmetro que modifica o valor da variável a1 na não-linearidade Tanh.

Parâmetros:

la1 Uma variável do tipo double contendo o valor da variável a1 para a não-linearidade Tanh.

F.6.3.8 virtual void Fastica::set_a2 (const double *la2*) [virtual]

Função com um parâmetro que modifica o valor da variável a2 na não-linearidade Gauss.

Parâmetros:

la2 Uma variável do tipo double contendo o valor da variável a2 para a não-linearidade Gauss.

F.6.3.9 virtual int Fastica::get_nonlinearity (void) const [virtual]

Função que retorna a não-linearidade que algoritmo está a utilizar.

Retorna:

Uma variável do tipo int contendo o tipo de não-linearidade (10 - Pow3; 20 - Tanh; 30 - Gauss).

F.6.3.10 virtual int Fastica::get_approach (void) const [virtual]

Função que retorna a aproximação que o algoritmo esta a utilizar.

Retorna:

Uma variável do tipo int contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

F.6.3.11 virtual int Fastica::get_maxiterations (void) const [virtual]

Função que retorna o número de iterações a que está regulado o algoritmo.

Retorna:

Uma variável do tipo int contendo o número de iterações.

F.6.3.12 virtual double Fastica::get_a1 (void) const [virtual]

Função que retorna o valor de a1 da não-linearidade Tanh.

Retorna:

Uma variável do tipo double contendo o valor de a1 da não-linearidade Tanh.

F.6.3.13 virtual double Fastica::get_a2 (void) const [virtual]

Função que retorna o valor de a2 da não-linearidade Gauss.

Retorna:

Uma variável do tipo double contendo o valor de a2 da não-linearidade Gauss.

F.6.3.14 virtual void Fastica::reset_fastica (void) [private, virtual]

Função que coloca os atributos da classe aos seus valores por defeito.

F.6.3.15 virtual void Fastica::init_fastica (void) [private, virtual]

Função que faz a inicialização dos atributos da classe.

F.6.4 Documentação dos dados membro

F.6.4.1 int Fastica::maxit [private]

Variável do tipo int contendo o número máximo de iterações.

F.6.4.2 int Fastica::g [private]

Variável do tipo int contendo a não-lineridade (10 - Pow3; 20 - Tanh; 30 - Gauss).

F.6.4.3 int Fastica::approach [private]

Variável do tipo int contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

F.6.4.4 mat Fastica::initguess [private]

Variável do tipo mat (it++) contendo a primeira estimativa da matriz de separação.

F.6.4.5 double Fastica::epsilon [private]

Variável do tipo double contendo o menor valor considerado.

F.6.4.6 int Fastica::a1 [private]

Variável do tipo int contendo o valor da variável a1 na não-linearidade Tanh.

F.6.4.7 int Fastica::a2 [private]

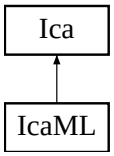
Variável do tipo int contendo o valor da variável a2 na não-linearidade Tanh.

F.7 Referência à classe IcaML

Sub-classe da classe **Ica**(p. 126) que implementa o algoritmo IcaML.

`#include <icapp.h>`

Diagrama de heranças da classe IcaML:



Membros públicos

- **IcaML** (const **Mix** &x)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 120).

- **IcaML** (char ∗lfilename, const int mixt, const int samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- virtual ~**IcaML** ()

  Destrutor.

- virtual void **separate** (void)

  Função que executa o algoritmo de ACI e calcula a matriz de separação.

- virtual void **set_maxiterations** (const int it)

  Função com um parâmetro que modifica o número de iterações do algoritmo.

- virtual int **get_maxiterations** (void) const

  Função que retorna o número de iterações a que está regulado o algoritmo.

- virtual vec **bic** (void)

Função que executa o algoritmo de Bic que estima o número de componentes independentes presentes nas misturas.

Membros privados

- virtual void **reset_icaml** (void)

  Função que coloca os atributos da classe aos seus valores por defeito.

- virtual void **init_icaml** (void)

  Função que faz a inicialização dos atributos da classe.

- virtual void **separate** (mat &S, mat &w, const mat &Xr)

  Função com três parâmetros que executa o algoritmo de ACI e calcula as matriz de separação e as componentes independentes.

Atributos Privados

- int **maxit**

  Variável do tipo int contendo o número máximo de iterações.

F.7.1 Descrição detalhada

Sub-classe da classe **Ica**(p. 126) que implementa o algoritmo IcaML.

F.7.2 Documentação dos Construtores & Destrutor

F.7.2.1 IcaML::IcaML (const Mix & *x*)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 120).

Parâmetros:

x Um objecto da classe **Mix**(p. 120) contendo as misturas.

F.7.2.2 IcaML::IcaML (char * *lfilename*, const int *mixt*, const int *samples*)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

lfilename Um ponteiro do tipo char apontando para o nome do ficheiro.

mixt Uma variável do tipo int contendo o número de misturas.

samples Uma variável do tipo int contento o número de amostras (instantes).

F.7.2.3 virtual IcaML::~IcaML () `[virtual]`

Destrutor.

F.7.3 Documentação dos métodos

F.7.3.1 virtual void IcaML::separate (void) `[virtual]`

Função que executa o algoritmo de ACI e calcula a matriz de separação.

F.7.3.2 virtual void IcaML::set_maxiterations (const int *it*) `[virtual]`

Função com um parâmetro que modifica o número de iterações do algoritmo.

Parâmetros:

it Uma variável do tipo int contendo o número de iterações.

F.7.3.3 virtual int IcaML::get_maxiterations (void) const `[virtual]`

Função que retorna o número de iterações a que está regulado o algoritmo.

Retorna:

Uma variável do tipo int contendo o número de iterações.

F.7.3.4 virtual vec IcaML::bic (void) `[virtual]`

Função que executa o algoritmo de Bic que estima o número de componentes independentes presentes nas misturas.

Retorna:

Uma variável (array) do tipo vec (it++) que contém a probabilidade de cada um dos modelos de número de componentes (de 2 até ao número máximo de misturas).

F.7.3.5 virtual void IcaML::reset_icaml (void) [private, virtual]

Função que coloca os atributos da classe aos seus valores por defeito.

F.7.3.6 virtual void IcaML::init_icaml (void) [private, virtual]

Função que faz a inicialização dos atributos da classe.

F.7.3.7 virtual void IcaML::separate (mat & *S*, mat & *w*, const mat & *Xr*) [private, virtual]

Função com três parâmetros que executa o algoritmo de ACI e calcula as matriz de separação e as componentes independentes.

Parâmetros:

S Um endereço do tipo mat (it++).

w Um endereço do tipo mat (it++) contendo os dados das misturas.

Xr Um endereço do tipo int contendo o número de misturas.

Retorna:

As componentes são retornadas em S, a matriz de separação é retornada em w e as misturas escalonadas são retornadas em Xr.

F.7.4 Documentação dos dados membro

F.7.4.1 int IcaML::maxit [private]

Variável do tipo int contendo o número máximo de iterações.

Anexo G: Documentação da biblioteca pyIca

G.1 Referência ao ficheiro pyIca.py

Componentes

- class **Mix**

  Classe que guarda os sinais misturados.

- class **Data**

  Classe que guarda os sinais misturados.

- class **Ica**

  Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

- class **Fastica**

  Sub-classe da classe **Ica**(p. 153) *que implementa o algoritmo Fastica.*

- class **IcaML**

  Sub-classe da classe **Ica**(p. 153) *que implementa o algoritmo IcaML.*

G.1.1 Descrição detalhada

Autor:

Michael Guerreiro

Data:

08/02/2006

G.2 Hierarquia de classes

Esta lista de heranças está organizada, dentro do possível, por ordem alfabética:

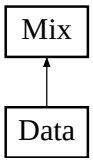
G.3 Lista de componentes

Lista de classes, estruturas, uniões e interfaces com uma breve descrição:

G.4 Referência à classe Mix

Super-Classe que guarda os sinais misturados.

Diagrama de heranças da classe Mix:



Membros públicos

- **__init__** ()

  Constructor base.

- **__init__** (mixclass)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe Mix ou **Data**(p. 151).

- **__init__** (filename, mixtures, samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- **load_matrix** (filename, mixtures, samples)

  Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

- numSamples **get_samples** ()

  Função que retorna o número de instantes.

- row **get_row** (i)

  Função com um parâmetro que retorna um vector contendo uma das misturas.

- col **get_col** (j)

  Função com um parâmetro que retorna um vector contento o valor de todas as misturas para um determinado instante.

- datamat **get_data** ()

  Função que retorna uma matriz contendo os dados das misturas.

- value **__getitem__** ()

  Função que permite o acesso aos valores indivíduais contidos na classe.

G.4.1 Descrição detalhada

Super-Classe que guarda os sinais misturados.

É utilizada como entrada nas funções da classe **Ica**(p. 153) e suas sub-classes.

G.4.2 Documentação dos métodos

G.4.2.1 Mix::__init__ ()

Constructor base.

Re-implementado em **Data** (p. 151).

G.4.2.2 Mix::__init__ (mixclass)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe Mix ou **Data**(p. 151).

Parâmetros:

mixclass Um objecto da classe Mix ou **Data**(p. 151) contendo as misturas.

Re-implementado em **Data** (p. 151).

G.4.2.3 Mix::__init__ (filename, mixtures, samples)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

Re-implementado em **Data** (p. 151).

G.4.2.4 Mix::load_matrix (filename, mixtures, samples)

Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

G.4.2.5 numSamples Mix::get_samples ()

Função que retorna o número de instantes.

Retorna:

Uma variável contendo o número de instantes.

G.4.2.6 row Mix::get_row (i)

Função com um parâmetro que retorna um vector contendo uma das misturas.

Parâmetros:

i Uma variável contendo o número da mistura.

Retorna:

Um vector numeric contendo uma mistura.

G.4.2.7 col Mix::get_col (j)

Função com um parâmetro que retorna um vector contento o valor de todas as misturas para um determinado instante.

Parâmetros:

i Uma variável contendo o número do instante.

Retorna:

Um vector numeric contendo o valor de todas as misturas para o instante i.

G.4.2.8 datamat Mix::get_data ()

Função que retorna uma matriz contendo os dados das misturas.

Retorna:

Uma matriz numeric contendo os dados das misturas.

G.4.2.9 value Mix::__getitem__ ()

Função que permite o acesso aos valores indivíduais contidos na classe.

O acesso aos dados é feito da forma [i][j], em que i e j representa a posição na matriz de dados.

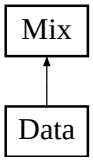
Retorna:

Uma variável contendo o valor da posição.

G.5 Referência à classe Data

Classe que guarda os sinais misturados.

Diagrama de heranças da classe Data:



Membros públicos

- **__init__** ()

  Constructor base.

- **__init__** (mixclass)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) *ou Data.*

- **__init__** (filename, mixtures, samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- **__init__** (mixtdata)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto numeric.

G.5.1 Descrição detalhada

Classe que guarda os sinais misturados.

É utilizada como entrada nas funções da classe **Ica**(p. 153) e suas sub-classes. Esta classe permite a introdução de matrizes numeric.

G.5.2 Documentação dos métodos

G.5.2.1 Data::__init__ ()

Constructor base.

Re-implementado de **Mix** (p. 147).

G.5.2.2 Data::__init__ (mixclass)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) ou Data.

Parâmetros:

mixclass Um objecto da classe **Mix**(p. 147) ou Data contendo as misturas.

Re-implementado de **Mix** (p. 147).

G.5.2.3 Data::__init__ (filename, mixtures, samples)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

Re-implementado de **Mix** (p. 147).

G.5.2.4 Data::__init__ (mixtdata)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto numeric.

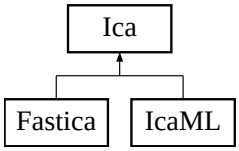
Parâmetros:

mixdata Uma matriz numeric contendo as misturas.

G.6 Referência à classe Ica

Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

Diagrama de heranças da classe Ica:



Membros públicos

- **__init__** ()

  Constructor base.

- **load_matrix** (filename, mixtures, samples)

  Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

- **calc_comp** ()

  Função que calcula as componentes a partir das misturas e da matriz de separação.

- **calc_icaproj** (comp)

  Função com um parâmetro que calcula as projecções de uma determinada componente.

- **set_ncomp** (ncomp)

  Função com um parâmetro que modifica o número de componentes a calcular.

- value **get_ncomp** ()

  Função que retorna o número de componentes.

- sepmat **get_sepmat** ()

  Função que retorna uma matriz contendo a matriz de separação.

- mixtmat **get_mixtmat** ()

Função que retorna uma matriz contendo a matriz misturas.

- projmat **get_proj** ()

  Função que retorna uma matriz contendo as projecções de uma determinada componente.

- compmat **get_comp** ()

  Função que retorna uma matriz contendo as componentes calculadas.

- datamat **get_data** ()

  Função que retorna uma matriz contendo os dados das misturas.

G.6.1 Descrição detalhada

Super-Classe que implementa as primitivas comuns aos algoritmos de ACI.

Esta classe serve de base para as classes que implementam os algortimos de ACI.

G.6.2 Documentação dos métodos

G.6.2.1 Ica::__init__ ()

Constructor base.

G.6.2.2 Ica::load_matrix (filename, mixtures, samples)

Função com três parâmetros que carrega os dados das misturas contidos num ficheiro para o objecto.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

G.6.2.3 Ica::calc_comp ()

Função que calcula as componentes a partir das misturas e da matriz de separação.

G.6.2.4 Ica::calc_icaproj (comp)

Função com um parâmetro que calcula as projecções de uma determinada componente.

Parâmetros:

comp Uma variável contendo o número da componente.

G.6.2.5 Ica::set_ncomp (ncomp)

Função com um parâmetro que modifica o número de componentes a calcular.

Parâmetros:

ncomp Uma variável contendo o número de componentes.

G.6.2.6 value Ica::get_ncomp ()

Função que retorna o número de componentes.

Retorna:

Uma variável contendo o número de componentes.

G.6.2.7 sepmat Ica::get_sepmat ()

Função que retorna uma matriz contendo a matriz de separação.

Retorna:

Uma matriz numeric contendo a matriz de separação.

G.6.2.8 mixtmat Ica::get_mixtmat ()

Função que retorna uma matriz contendo a matriz misturas.

Retorna:

Uma matriz numeric contendo a matriz de misturas.

G.6.2.9 projmat Ica::get_proj ()

Função que retorna uma matriz contendo as projecções de uma determinada componente.

Retorna:

Uma matriz numeric contendo as projecções de uma determinada componente.

G.6.2.10 compmat Ica::get_comp ()

Função que retorna uma matriz contendo as componentes calculadas.

Retorna:

Uma matriz numeric contendo as componentes calculadas.

G.6.2.11 datamat Ica::get_data ()

Função que retorna uma matriz contendo os dados das misturas.

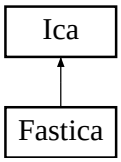
Retorna:

Uma matriz numeric contendo os dados das misturas.

G.7 Referência à classe Fastica

Sub-classe da classe **Ica**(p. 153) que implementa o algoritmo Fastica.

Diagrama de heranças da classe Fastica:



Membros públicos

- **__init__** (mixclass)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) *ou* **Data**(p. 151).

- **__init__** (filename, mixtures, samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- **separate** ()

  Função que executa o algoritmo de ACI e calcula a matriz de separação.

- **set_approach** (approach)

  Função com um parâmetro que modifica o tipo de aproximação do algoritmo.

- **set_nonlinearity** (g)

  Função com um parâmetro que modifica a não-linearidade que o algoritmo vai utilizar.

- **set_a1** (a1)

  Função com um parâmetro que modifica o valor da variável a1 na não-linearidade Tanh.

- **set_a2** (a2)

  Função com um parâmetro que modifica o valor da variável a2 na não-linearidade Gauss.

- approach **get_approach** ()

  Função que retorna a aproximação que o algoritmo esta a utilizar.

- g **get_nonlinearity** ()

  Função que retorna a não-linearidade que algoritmo está a utilizar.

- iterations **get_maxiterations** ()

  Função que retorna o número de iterações a que está regulado o algoritmo.

- a1 **get_a1** ()

  Função que retorna o valor de a1 da não-linearidade Tanh.

- a2 **get_a2** ()

  Função que retorna o valor de a2 da não-linearidade Gauss.

G.7.1 Descrição detalhada

Sub-classe da classe **Ica**(p. 153) que implementa o algoritmo Fastica.

G.7.2 Documentação dos métodos

G.7.2.1 Fastica::__init__ (mixclass)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) ou **Data**(p. 151).

Parâmetros:

mixclass Um objecto da classe **Mix**(p. 147) ou **Data**(p. 151) contendo as misturas.

G.7.2.2 Fastica::__init__ (filename, mixtures, samples)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

G.7.2.3 Fastica::separate ()

Função que executa o algoritmo de ACI e calcula a matriz de separação.

G.7.2.4 Fastica::set_approach (approach)

Função com um parâmetro que modifica o tipo de aproximação do algoritmo.

Parâmetros:

method Uma variável contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

G.7.2.5 Fastica::set_nonlinearity (g)

Função com um parâmetro que modifica a não-linearidade que o algoritmo vai utilizar.

Parâmetros:

g Uma variável contendo o tipo de não-linearidade (10 - Pow3; 20 - Tanh; 30 - Gauss).

G.7.2.6 Fastica::set_a1 (a1)

Função com um parâmetro que modifica o valor da variável a1 na não-linearidade Tanh.

Parâmetros:

a1 Uma variável contendo o valor de a1 para a não-linearidade Tanh.

G.7.2.7 Fastica::set_a2 (a2)

Função com um parâmetro que modifica o valor da variável a2 na não-linearidade Gauss.

Parâmetros:

a2 Uma variável contendo o valor de a2 para a não-linearidade Gauss.

G.7.2.8 approach Fastica::get_approach ()

Função que retorna a aproximação que o algoritmo esta a utilizar.

Retorna:

Uma variável contendo o tipo de aproximação (1 - Simétrica; 2 - Deflação).

G.7.2.9 g Fastica::get_nonlinearity ()

Função que retorna a não-linearidade que algoritmo está a utilizar.

Retorna:

Uma variável contendo o tipo de não-linearidade (10 - Pow3; 20 - Tanh; 30 - Gauss).

G.7.2.10 iterations Fastica::get_maxiterations ()

Função que retorna o número de iterações a que está regulado o algoritmo.

Retorna:

Uma variável contendo o número de iterações.

G.7.2.11 a1 Fastica::get_a1 ()

Função que retorna o valor de a1 da não-linearidade Tanh.

Retorna:

Uma variável contendo o valor de a1 da não-linearidade Tanh.

G.7.2.12 a2 Fastica::get_a2 ()

Função que retorna o valor de a2 da não-linearidade Gauss.

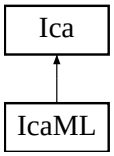
Retorna:

Uma variável contendo o valor de a2 da não-linearidade Gauss.

G.8 Referência à classe IcaML

Sub-classe da classe **Ica**(p. 153) que implementa o algoritmo IcaML.

Diagrama de heranças da classe IcaML:



Membros públicos

- **__init__** (mixclass)

  Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) *ou* **Data**(p. 151).

- **__init__** (filename, mixtures, samples)

  Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

- **separate** ()

  Função que executa o algoritmo de ACI e calcula a matriz de separação.

- probvec **bic** ()

  Função que executa o algoritmo de Bic que estima o número de componentes independentes presentes nas misturas.

- **set_maxiterations** (iterations)

  Função com um parâmetro que modifica o número de iterações do algoritmo.

- iterations **get_maxiterations** ()

  Função que retorna o número de iterações a que está regulado o algoritmo.

G.8.1 Descrição detalhada

Sub-classe da classe **Ica**(p. 153) que implementa o algoritmo IcaML.

G.8.2 Documentação dos métodos

G.8.2.1 IcaML::__init__ (mixclass)

Construtor com um parâmetro que cria um objecto utilizando os dados de um objecto da classe **Mix**(p. 147) ou **Data**(p. 151).

Parâmetros:

mixclass Um objecto da classe **Mix**(p. 147) ou **Data**(p. 151) contendo as misturas.

G.8.2.2 IcaML::__init__ (filename, mixtures, samples)

Construtor com três parâmetros que cria um objecto utilizando os dados contidos num ficheiro.

Parâmetros:

filename Uma contendo o nome do ficheiro.

mixtures Uma variável contendo o número de misturas.

samples Uma variável contento o número de amostras (instantes).

G.8.2.3 IcaML::separate ()

Função que executa o algoritmo de ACI e calcula a matriz de separação.

G.8.2.4 probvec IcaML::bic ()

Função que executa o algoritmo de Bic que estima o número de componentes independentes presentes nas misturas.

Retorna:

Um vector numeric que contém a probabilidade de cada um dos modelos de número de componentes (de 2 até ao número máximo de misturas).

G.8.2.5 IcaML::set_maxiterations (iterations)

Função com um parâmetro que modifica o número de iterações do algoritmo.

Parâmetros:

iterations Uma variável contendo o número de iterações.

G.8.2.6 iterations IcaML::get_maxiterations ()

Função que retorna o número de iterações a que está regulado o algoritmo.

Retorna:

Uma variável contendo o número de iterações.

Anexo H: Listagem do Código C++ do Módulo pyIca

```
1  /* Ficheiro de código ica.h da livraria para ACI desenvolvido por
       Michael Guerreiro */
2
3  #ifndef ICA_H
4  #define ICA_H
5
6  #include <stdlib.h>
7  #include "itbase.h"
8  #include <iostream>
9  #include <fstream>
10 #include <cmath>
11
12 /* Constantes para variáveis referentes ao algoritmo Fastica */
13
14 #define FICA_APPROACH_SYMM 1
15 #define FICA_APPROACH_DEFL 2
16
17 #define FICA_NONLIN_POW3 10
18 #define FICA_NONLIN_TANH 20
19 #define FICA_NONLIN_GAUSS 30
20
21 using namespace std;
22
23 /* Classe Mix para armazenamento dos dados misturados */
24 class Mix
25 {
26         protected:
27                 int m, s;
28                 char *filename;
29
30         public:
31                 mat matrix;
32
33         public:
34                 Mix();
35                 Mix(const Mix &other);
36                 Mix(const mat &mtx);
37
38                 Mix(char *lfilename, const int mixt, const int samples)
                        ;
39
40                 virtual ~Mix();
41
```

```
42                  virtual void load_matrix(char *lfilename, const int
                        mixt, const int samples);
43                  virtual void save_matrix(char *lfilename);
44                  virtual int get_samples(void) const;
45                  virtual int get_mixt(void) const;
46                  virtual vec get_row(int i) const;
47                  virtual vec get_col(int i) const;
48                  virtual mat get_data(void) const;
49                  virtual char *get_filename(void) const;
50
51          protected:
52                  virtual void reset(void);
53                  virtual void set_matrix(const mat &mtx);
54
55
56  };
57
58  /* Super-Classe ICA que define as funções características de todos os
        algoritmos ICA */
59
60  class Ica
61  {
62          protected:
63                  int n;
64
65                  Mix *X;
66                  mat *W;
67                  mat *A;
68                  mat *Y;
69                  mat *ICAproj;
70
71          public:
72                  Ica();
73                  virtual ~Ica();
74
75                  virtual void load_matrix(char *lfilename, const int
                        mixt, const int samples);
76                  virtual void set_ncomp(int ncomp);
77                  virtual int get_ncomp(void) const;
78                  virtual void calc_comp(void);
79                  virtual void calc_icaproj(const int comp);
80                  virtual mat get_sepmat(void);
81                  virtual mat get_mixtmat(void);
82                  virtual mat get_comp(void);
83                  virtual mat get_proj(void);
84                  virtual mat get_data(void);
85
86          protected:
87                  virtual void reset(void);
88                  virtual void rmmean(mat &Xm, const mat &Xv);
89                  virtual double scale_x(mat &Xs, const mat &Xm);
90                  virtual void pca(mat &Wh, const mat &Xm, const int &
                        mixt, const int &samples);
91
```

```
92  };
93
94  /* Sub-Classe Fastica que implementa o algoritmo Fastica */
95
96  class Fastica : public Ica
97  {
98          private:
99                  int maxit, g, approach;
100                 mat initguess;
101                 double epsilon, a1, a2;
102
103         public:
104                 Fastica(const Mix &x);
105                 Fastica(char *lfilename, const int mixt, const int
                        samples);
106
107                 virtual ~Fastica();
108
109                 virtual void separate(void);
110                 virtual void set_maxiterations(const int it);
111                 virtual void set_nonlinearity(const int ig);
112                 virtual void set_approach(const int method);
113                 virtual void set_initguess(char *matrix);
114                 virtual void reset_initguess();
115                 virtual void set_a1(const double la1);
116                 virtual void set_a2(const double la2);
117
118                 virtual int get_nonlinearity(void) const;
119                 virtual int get_approach(void) const;
120                 virtual int get_maxiterations(void) const;
121                 virtual double get_a1(void) const;
122                 virtual double get_a2(void) const;
123
124         private:
125                 virtual void reset_fastica(void);
126
127 };
128
129 /* Sub-Classe IcaML que implementa o algoritmo IcaML */
130
131 class IcaML : public Ica
132 {
133         private:
134                 int maxit;
135
136         public:
137                 IcaML(const Mix &x);
138                 IcaML(char *lfilename, const int mixt, const int
                        samples);
139
140                 virtual ~IcaML();
141
142                 virtual void separate(void);
143                 virtual void set_maxiterations(const int it);
```

```
144                     virtual int get_maxiterations(void) const;
145                     virtual vec bic(void);
146
147             private:
148                     virtual void reset_icaml(void);
149                     virtual void separate(mat &S, mat &w, const mat &Xr);
150     };
151
152     #endif
153
154
155     /*Ficheiro de código ica.cxx desenvolvido por Michael Guerreiro*/
156
157     #include "ica.h"
158
159     //-------------------------------------------------------------//
160
161     /* Estrutura auxiliar para os dados */
162
163     struct param
164     {
165             mat X;
166             int M;
167             int N;
168     };
169
170     /* Funções auxiliares para as classes */
171
172     static void sum_w_hT_h_wT(mat &m, vec &v1, vec &v2);
173     static vec sumcol(const mat A);
174     static mat mpower(const mat A, const double y);
175     static mat orth(const mat A);
176     static vec &operator << (vec &os, const double value);
177     static vec &operator << (vec &os, const vec v1);
178     static fstream &operator << (fstream &file, const mat &m);
179     static vec diff(vec &v1);
180     static vec diff(vec v1);
181     static double norm_inf(const vec &v1);
182     static void ica_mlf(double &f, vec &dW, const param &par, const vec &W)
        ;
183     static double interpolate(mat &xfd, int &n);
184     static void softline(double &alpha, double &F, vec &g, int &neval,
185     double &slrat, const param &par, const vec &x, const vec &h);
186     static bool check(double &F, vec &g, int &n, param &par, vec &x, vec &
        opts);
187     static void ucminf(vec &info, param &par, vec &x, vec &opts);
188     static mat pinv(mat W);
189
190     //-------------------------------------------------------------//
191
192     /* Construtor simples da classe Mix */
193     Mix::Mix()
194     {
195             reset();
```

```
196 }
197
198 /* Construtor cópia da classe Mix */
199 Mix::Mix(const Mix &other)
200 {
201         this-> matrix = other.matrix;
202         this-> m = other.get_mixt();
203         this-> s = other.get_samples();
204         this-> filename = other.get_filename();
205 }
206
207 /* Construtor da classe Mix com uma matrix como entrada */
208 Mix::Mix(const mat &mtx)
209 {
210         set_matrix(mtx);
211 }
212
213 /* Construtor da classe Mix com ficheiro como entrada */
214 Mix::Mix (char *lfilename, const int mixt, const int samples)
215 {
216         load_matrix(lfilename, mixt, samples);
217 }
218
219 /* Retorna número de amostras (colunas) da matriz de dados da classe
    Mix */
220 int Mix::get_samples(void) const
221 {
222         return s;
223 }
224
225 /* Retorna número de misturas (linhas) da matriz de dados da classe Mix
     */
226 int Mix::get_mixt(void) const
227 {
228         return m;
229 }
230
231 /* Retorna linha da matriz de dados da classe Mix */
232 vec Mix::get_row(int i) const
233 {
234         vec a;
235         if ((i > m-1) || (i < 0)) return a;
236         return matrix.get_row(i);
237 }
238
239 /* Retorna coluna da matriz de dados da classe Mix */
240 vec Mix::get_col(int i) const
241 {
242         vec a;
243         if ((i > s-1) || (i < 0)) return a;
244         return matrix.get_col(i);
245 }
246
247 /* Retorna matriz de dados da classe Mix */
```

```
248 mat Mix::get_data(void) const
249 {
250         return matrix;
251 }
252
253 /* Retorna nome do ficheiro da classe Mix */
254 char *Mix::get_filename(void) const
255 {
256         return filename;
257 }
258
259 /* Destrutor da classe Mix */
260 Mix::~Mix()
261 {
262
263 }
264
265 /* Construtor da classe Mix */
266 void Mix::set_matrix(const mat &mtx)
267 {
268         this-> matrix = mtx;
269         this-> m = mtx.rows();
270         this-> s = mtx.cols();
271         this-> filename = "";
272 }
273
274 /* Carrega matrix a partir de um ficheiro para classe Mix */
275 void Mix::load_matrix(char *lfilename, const int mixt, const int
    samples)
276 {
277         std::fstream file;
278
279         file.open(lfilename,std::ios::in);
280
281         if (!file.is_open())
282         {
283                 std::cout << "Não consegue abrir ficheiro!" << endl;
284         }
285
286         else
287         {
288                 std::stringbuf *buf = new std::stringbuf();
289                 std::string test;
290
291                 while (!file.eof())
292                 {
293                         if (!file.get(*buf,'\n')) break;
294                         file.ignore(1,'\n');
295                         buf->sputc(';');
296                 }
297
298                 file.close();
299
300                 test = buf->str();
```

```
301                     delete buf;
302                     buf = 0;
303
304                     test.erase(test.length()-1);
305                     this -> matrix.set(test.c_str());
306             }
307
308             if ( (matrix.rows() == mixt) && (matrix.cols() == samples) )
309             {
310                     this-> m = mixt;
311                     this-> s = samples;
312                     this-> filename = lfilename;
313             }
314             else
315             {
316                     matrix.set_size(0,0);
317                     reset();
318                     std::cout << "Tamanho de matriz incorrecta!" << endl;
319             }
320
321
322     }
323
324     /* Salva matriz de dados da classe Mix para um ficheiro */
325     void Mix::save_matrix(char *lfilename)
326     {
327             fstream file;
328
329             file.open(lfilename,std::ios::out);
330             file << matrix;
331             file.close();
332             this-> filename = lfilename;
333     }
334
335     /* Coloca as variáveis da classe Mix a zero */
336     void Mix::reset(void)
337     {
338             this-> m = 0;
339             this-> s = 0;
340             this-> filename = "";
341     }
342
343     //-------------------------------------------------------//
344
345     /* Construtor simples da classe Ica */
346     Ica::Ica()
347     {
348             reset();
349     }
350
351     /* Destrutor da classe Ica */
352     Ica::~Ica()
353     {
354
```

```
355 }
356
357 /* Construtor da classe Ica utilizando um ficheiro */
358 void Ica::load_matrix(char *lfilename, const int mixt, const int
    samples)
359 {
360         X = new Mix(lfilename, mixt, samples);
361         this -> n = X->get_mixt();
362 }
363
364 /* Retorna o número de componentes da classe Ica */
365 int Ica::get_ncomp(void) const
366 {
367         return n;
368 }
369
370 /* Modifica o número de componentes da classe Ica */
371 void Ica::set_ncomp(const int ncomp)
372 {
373         this-> n = ncomp;
374 }
375
376 /* Coloca as variáveis da classe Ica a zero */
377 void Ica::reset(void)
378 {
379         this-> n = 0;
380 }
381
382 /* Centra os dados da matriz de dados da classe Ica */
383 void Ica::rmmean(mat &Xm, const mat &Xv)
384 {
385         Xm = zeros( Xv.rows(), Xv.cols() );
386         vec md = zeros( Xv.rows() );
387
388         for ( int i= 0; i< Xv.rows(); i++ )
389         {
390                 md(i) = mean( Xv.get_row(i) );
391
392                 for ( int j= 0; j< Xv.cols(); j++ )
393                 {
394                         Xm(i,j) = Xv(i,j)-md(i);
395                 }
396         }
397 }
398
399 /* Escalonamento dos dados da matriz de dados da classe Ica */
400 double Ica::scale_x(mat &Xs, const mat &Xm)
401 {
402         vec vm;
403         vm << abs(max(Xm)) << abs(min(Xm));
404         double scale = max(vm);
405         Xs = Xm / scale;
406
407         return scale;
```

```
408 }
409
410 /* Calculo das componentes a partir da matriz de separação da classe
        Ica */
411 void Ica::calc_comp(void)
412 {
413         if(!X)
414         {
415                 std::cout << "Matriz não carregada!" << endl;
416                 return;
417         }
418
419         if ((X->matrix.rows()<=1) || (X->matrix.cols()<=1))
420         {
421                 std::cout << "Matriz não carregada ou inválida!" <<
                        endl;
422                 return;
423         }
424
425         if(!W)
426         {
427                 std::cout << "Matriz de separação W não calculada!" <<
                        endl;
428                 return;
429         }
430
431         if ((W->rows()<=1) || (W->cols()<=1))
432         {
433                 std::cout << "Matriz de separação W não carregada ou
                        inválida!" << endl;
434                 return;
435         }
436
437         if(!this->Y) this->Y = new mat ((*W) * X->matrix);
438         else (*Y) = (*W) * X->matrix;
439 }
440
441 /* Calculo das projecções a partir de uma das componentes calculadas da
        classe Ica */
442 void Ica::calc_icaproj(const int comp)
443 {
444         if(!X)
445         {
446                 std::cout << "Matriz não carregada!" << endl;
447                 return;
448         }
449
450         if ((X->matrix.rows()<=1) || (X->matrix.cols()<=1))
451         {
452                 std::cout << "Matriz não carregada ou inválida!" <<
                        endl;
453                 return;
454         }
455
```

```
456             if(!W)
457             {
458                     std::cout << "Matriz de separação W não calculada!" <<
                            endl;
459                     return;
460             }
461
462             if ((W->rows()<=1) || (W->cols()<=1))
463             {
464                     std::cout << "Matriz de separação W não carregada ou
                            inválida!" << endl;
465                     return;
466             }
467
468             if (X->get_mixt()==n) this-> A = new mat(inv(*W));
469             else this-> A = new mat(pinv(*W));
470
471             vec act;
472             mat temp;
473
474             act = transpose(X->matrix) * (W->get_row(comp));
475             temp = outer_product(A->get_col(comp),act);
476             cout << A->get_col(comp)<< endl;
477             cout << temp.get_col(0) << endl;
478
479             if (!this->ICAproj) this->ICAproj = new mat(temp);
480             else (*ICAproj) = temp;
481     }
482
483     /* Calculo da análise de das componetes principais da classe Ica */
484     void Ica::pca(mat &Wh, const mat &Xm, const int &mixt, const int &
            samples)
485     {
486             vec D;
487             mat U;
488             int ilim = mixt - n;
489             int slim = mixt - 1;
490
491
492             eig_sym((Xm*transpose(Xm))/samples, D, U);
493
494             ivec k = sort_index(D);
495
496             Wh = diag(1.0/sqrt(D(ilim,slim))) * transpose(U.get_cols(k(ilim
                    ,slim)));
497     }
498
499     /* Retorna a matriz de separação da classe Ica */
500     mat Ica::get_sepmat(void)
501     {
502             mat w;
503             if (!W) return w;
504             else return (*W);
505
```

```
506 }
507
508 /* Retorna a matriz de mistura da classe Ica */
509 mat Ica::get_mixtmat(void)
510 {
511         mat a;
512         if (!A) return a;
513         else return (*A);
514 }
515
516 /* Retorna a matriz de componentes da classe Ica */
517 mat Ica::get_comp(void)
518 {
519         mat y;
520         if (!Y) return y;
521         else return (*Y);
522
523 }
524
525 /* Retorna a matriz de projecções da classe Ica */
526 mat Ica::get_proj(void)
527 {
528         mat icaproj;
529         if (!ICAproj) return icaproj;
530         else return (*ICAproj);
531 }
532
533 /* Retorna a matriz de misturas (dados originais) da classe Ica */
534 mat Ica::get_data(void)
535 {       mat x;
536         if (!X) return x;
537         else return X->matrix;
538
539 }
540
541 //-------------------------------------------------------//
542
543 /* Construtor da classe Fastica através da introdução de um objecto do
      tipo Mix */
544 Fastica::Fastica(const Mix &x)
545 {
546         this->X = new Mix(x);
547         if (X)
548         {
549                 this-> n = X->get_mixt();
550         }
551         else reset_fastica();
552 }
553
554 /* Construtor da classe Fastica através da introdução de um ficheiro */
555 Fastica::Fastica(char *lfilename, const int mixt, const int samples)
556 {
557         load_matrix(lfilename, mixt, samples);
558         reset_fastica();
```

```
559 }
560
561 /* Separação das componentes */
562 void Fastica::separate(void)
563 {
564         if(!X)
565         {
566                 std::cout << "Matriz não carregada!" << endl;
567                 return;
568         }
569
570         if ((X->matrix.rows()<=1) || (X->matrix.cols()<=1))
571         {
572                 std::cout << "Matriz não carregada ou inválida!" <<
                        endl;
573                 return;
574         }
575
576         if ((n > X->get_mixt()) || (n < 2))
577         {
578              std::cout << "Número de componentes não pode ser superior
                    ao número de misturas e não inferior a dois!" << endl;
579                 return;
580         }
581
582         mat Xm, Wh;
583         int m = X->get_mixt(), s = X->get_samples(),failureLimit= 5,
                numFailures=0;
584
585         //.........Pré-Processamento.........//
586
587         rmmean(Xm, X->matrix);
588         pca(Wh, Xm, m, s);
589         Xm = Wh*Xm;
590
591         //.........Processamento Simétrico.........//
592         if (approach == FICA_APPROACH_SYMM)
593         {
594                 RNG_randomize ();
595
596                 double minAbsCos = 0.0;
597                 mat B = zeros(n, n);
598
599                 if ((initguess.rows() == m) && (initguess.cols() == n))
600                         B = Wh*initguess;
601
602                 else if ((initguess.rows() == 0) && (initguess.cols()
                        == 0))
603                         B = orth(randu(n, n) - 0.5);
604                 else
605                 {
606                         cout << "Matriz inicial com tamanho incorrecto
                                !" << endl;
607                         B = orth(randu(n, n) - 0.5);
```

```
608                 }
609
610                 mat BOld(B.rows(),B.cols());
611
612                 for (int rd = 0; rd < maxit; rd++)
613                 {
614                         if (rd == (maxit-1))
615                         {
616                                 cout << "Algoritmo não convergiu!" <<
                                        endl;
617                                 return;
618                         }
619
620                         B = B * mpower(transpose(B) * B , -0.5);
621                         minAbsCos = min(abs(diag(transpose(B) * BOld)))
                                ;
622
623                         if ((1-minAbsCos) < epsilon)
624                         {
625                                 break;
626                         }
627
628                         BOld = B;
629
630                         switch (g)
631                         {
632                                 case FICA_NONLIN_POW3:
633                                 {
634                                         B = (Xm*pow(transpose(Xm)*B, 3)
                                                )/s - 3*B;
635                                         break;
636                                 }
637
638                                 case FICA_NONLIN_TANH:
639                                 {
640                                         mat hypTan = tanh( a1*transpose
                                                (Xm)*B );
641         B = (Xm*hypTan)/s - elem_mult(reshape(repeat(sumcol(1-pow(
                hypTan, 2)), B.rows()), B.rows(), B.cols()), B)/s * a1;
642                                         break;
643
644                                 }
645
646                                 case FICA_NONLIN_GAUSS:
647                                 {
648                                         mat u = transpose(Xm)*B;
649                                         mat U = pow(u, 2);
650                                         mat exU = exp(-a2*U/2);
651                                         mat gauss = elem_mult(u, exU);
652                                         mat Gauss = elem_mult (1-a2*U,
                                                exU);
653                         B = (Xm*gauss)/s - elem_mult(reshape(repeat(
                                sumcol(Gauss), B.rows()), B.rows(), B.cols()
                                ), B)/s;
```

```
654                                         break;
655
656                                 }
657                         }
658                 }
659                 if (!this->W) this-> W = new mat(transpose(B)*Wh);
660                 else (*W) = transpose(B)*Wh;
661         }
662         //.........Processamento Deflacção.........//
663         else if (approach == FICA_APPROACH_DEFL)
664         {
665                 RNG_randomize();
666                 mat B = zeros(n, n), Wt=zeros(n,m);
667
668                 int round = 1;
669                 int numFailures = 0;
670                 vec w, wOld(n);
671
672                 while (round <= n)
673                 {
674                         if ((initguess.rows() == m) && (initguess.cols
                            () == n))
675                                 w = Wh*initguess.get_col(round);
676
677                         else
678                                 w = randu(n) - 0.5;
679
680                         w -= B * transpose(B)*w;
681                         w /= norm(w);
682
683
684                         wOld.zeros();
685
686                         int i= 1;
687
688                         while (i<= maxit+1)
689                         {
690
691                                 w -= B * transpose(B)*w;
692                                 w /= norm(w);
693
694                                 if (i==maxit+1)
695                                 {
696                                         round--;
697                                         numFailures++;
698                                         if (numFailures > failureLimit)
699                                         {
700                                                 cout << "Componente " <<
                                                    round+1 << " não
                                                    convergiu!" << endl;
701                                                 if (round ==0)
702                                                 {
703                                                         Wt = transpose(
                                                            B)*Wh;
```

```
704                                     }
705                                     return;
706                             }
707                             break;
708                     }
709
710                     if ((norm(w-wOld) < epsilon) || (norm(w
                            +wOld) < epsilon) )
711                     {
712
713                             numFailures = 0;
714                             B.set_col(round-1, w);
715                             Wt.set_row(round-1, transpose(
                                    Wh)*w );
716                             break;
717                     }
718
719                     wOld = w;
720
721                     switch ( g ) {
722
723                             case FICA_NONLIN_POW3 :
724                             {
725                                     w = (Xm*pow(transpose(
                                            Xm)*w, 3))/s - 3*w;
726                                     break;
727                             }
728                             case FICA_NONLIN_TANH :
729                             {
730                                     vec hyperTan = tanh(a1*
                                            transpose(Xm)*w);
731                                     w = ((Xm*hyperTan)-(a1*
                                            sum(1-pow(hyperTan
                                            ,2))*w))/s;
732                                     break;
733                             }
734
735                             case FICA_NONLIN_GAUSS :
736                             {
737                                     vec u = transpose(Xm)*w
                                            ;
738                                     vec U = pow(u, 2);
739                                     vec exU = exp( -a2*U/2
                                            );
740                                     vec gauss = elem_mult(u
                                            , exU);
741                                     vec Gauss = elem_mult
                                            ((1-a2*U), exU);
742                                     w = ((Xm*gauss)-(sum(
                                            Gauss)*w))/s;
743                                     break;
744                             }
745                     }
746
```

```
747                                     w /= norm(w);
748                                     i++;
749                             }
750                             round++;
751                     }
752                     if (!this->W) this-> W = new mat(Wt);
753                     else (*W) = Wt;
754             }
755
756     }
757
758     /* Definir número de iterações */
759     void Fastica::set_maxiterations(const int it)
760     {
761             this->maxit = it;
762     }
763
764     /* Definir não-linearidade */
765     void Fastica::set_nonlinearity(const int ig)
766     {
767             this->g = ig;
768     }
769
770     /* Definir aproximação (Simétrico, Deflacção) */
771     void Fastica::set_approach(const int method)
772     {
773             this->approach = method;
774     }
775
776     /* Definir a1 para não lineridade tanh */
777     void Fastica::set_a1(const double la1)
778     {
779             this->a1 = la1;
780     }
781
782     /* Definir a1 para não lineridade tanh */
783     void Fastica::set_a2(const double la2)
784     {
785             this->a2 = la2;
786     }
787
788     /* Definir aproximação da matriz de separação inicial */
789     void Fastica::set_initguess(char *matrix)
790     {
791             this->initguess = matrix;
792     }
793
794     /* Reset da aproximação da matriz de separação inicial */
795     void Fastica::reset_initguess()
796     {
797             initguess.set_size(0,0);
798     }
799
800     /* Retorna número de iterações */
```

```
801 int Fastica::get_maxiterations(void) const
802 {
803         return maxit;
804 }
805
806 /* Retorna não-linearidade */
807 int Fastica::get_nonlinearity(void) const
808 {
809         return g;
810 }
811
812 /* Retorno da aproximação (Simétrico, Deflacção) */
813 int Fastica::get_approach(void) const
814 {
815         return approach;
816 }
817
818 /* Retorno de a1 para não lineridade tanh */
819 double Fastica::get_a1(void) const
820 {
821         return a1;
822 }
823
824 /* Retorno de a2 para não lineridade gauss */
825 double Fastica::get_a2(void) const
826 {
827         return a2;
828 }
829
830 /* Reset das variáveis da classe Fastica */
831 void Fastica::reset_fastica(void)
832 {
833         this-> approach = FICA_APPROACH_DEFL;
834         this-> g = FICA_NONLIN_GAUSS;
835         this-> epsilon = 0.0001;
836         this-> maxit = 10000;
837         this-> a1 = 1.0;
838         this-> a2 = 1.0;
839 }
840
841 /* Destrutor da classe Fastica */
842 Fastica::~Fastica()
843 {
844
845 }
846
847 //-----------------------------------------------//
848
849 /* Construtor da classe IcaML através da introdução de um objecto do
       tipo Mix */
850 IcaML::IcaML(const Mix &x)
851 {
852         this->X = new Mix(x);
853         if (X)
```

```
854             {
855                     this-> n = X->get_mixt();
856             }
857             else reset_icaml();
858     }
859
860     /* Construtor da classe IcaML através de um ficheiro */
861     IcaML::IcaML(char *lfilename, const int mixt, const int samples)
862     {
863             load_matrix(lfilename, mixt, samples);
864             reset_icaml();
865     }
866
867     /*Cálculo do número de componentes através do método BIC*/
868     vec IcaML::bic(void)
869     {
870             tic();
871             vec P;
872
873             if(!X)
874             {
875                     std::cout << "Matriz não carregada!" << endl;
876                     return P;
877             }
878
879             if ((X->matrix.rows()<=1) || (X->matrix.cols()<=1))
880             {
881                     std::cout << "Matriz não carregadaou inválida!" << endl
                            ;
882                     return P;
883             }
884
885             int m = X->get_mixt();
886             int s = X->get_samples();
887             mat U,V,D,Xm,S,Wt,V_UD;
888             vec d, logP;
889             ivec K;
890             int k;
891
892             if (s > m)
893             {
894                     svd(X->matrix.T(), V, d, U);
895                     V = V(0,-1,0,m-1);
896             }
897
898             else
899             {
900                     svd(X->matrix, U, d, V);
901                     if (m>s) U = U(0,-1,0,s-1);
902             }
903
904             D = diag(d);
905
906             int M = U.rows();
```

```
907             int N = V.rows();
908             double sigma2, dim;
909
910             Xm = D*transpose(V);
911
912             stringstream ss;
913             string lim;
914
915             ss << m;
916             ss >> lim;
917
918             K.set("2:"+lim);
919             char t;
920
921             for(int i = 0; i < K.size(); i++)
922             {
923                     k=K(i);
924                     //cout << "Dim: " << k << endl;
925
926                     separate(S,Wt,Xm(0,k-1,0,-1));
927                     logP << N*log(abs(det(Wt))) - sum(sum(log(cosh(S)))) -
                            N*k*log(pi);
928
929                     if (k<M)
930                     {
931                             V_UD = U(0,-1,k,-1)*D(k,-1,k,-1);
932                             sigma2 = sum(sum(pow(V_UD,2)))/((M-k)*N);
933                             logP(i) -= 0.5*(M-k)*N*(log(sigma2) +1);
934                     }
935                     dim = k*(2*M-k+1)/2 + 1 + k*k;
936                     logP(i) -= 0.5*dim*log(N);
937             }
938
939             P = exp((logP-max(logP))/N);
940             P = P/sum(P);
941             toc_print();
942             return P;
943     }
944
945     /* Modo de separação das componentes simples da classe IcaML*/
946     void IcaML::separate(void)
947     {       tic();
948             if(!X)
949             {
950                     std::cout << "Matriz não carregada!" << endl;
951                     return;
952             }
953
954             if ((X->matrix.rows()<=1) || (X->matrix.cols()<=1))
955             {
956                     std::cout << "Matriz não carregadaou inválida!" << endl
                            ;
957                     return;
958             }
```

```
959
960             if ((n > X->get_mixt()) || (n < 2))
961             {
962                 std::cout << "Número de componentes não pode ser superior
                        ao número de misturas e não inferior a dois!" << endl;
963                 return;
964             }
965
966
967             mat Xm,Wh;
968             int m = X->get_mixt();
969             int s = X->get_samples();
970             param par;
971
972             double scaleX = scale_x(Xm, X->matrix);
973
974             pca(Wh, Xm, m, s);
975
976             par.X = Wh * Xm;
977             par.M = n;
978             par.N = s;
979
980             vec info;
981             vec Wt = cvectorize(eye(n));
982
983             vec opts = "1 1e-4 1e-8";
984             opts << maxit;
985
986             ucminf(info, par, Wt, opts);
987
988             if (info(3)==maxit) cout << "Algoritmo não convergiu!" << endl;
989
990             if (!this->W) this->W = new mat((reshape(Wt,n,n)*Wh)/scaleX);
991             else (*W) = (reshape(Wt,n,n)*Wh)/scaleX;
992             toc_print();
993     }
994
995     /* Modo de separação das componentes para cálculo do bic */
996     void IcaML::separate(mat &S, mat &w, const mat &Xr)
997     {
998             int tn = n;
999
1000            mat Xm,Wh;
1001            int m = Xr.rows();
1002            int s = Xr.cols();
1003            set_ncomp(m);
1004
1005            param par;
1006            double scaleX = scale_x(Xm, Xr);
1007            pca(Wh, Xm, m, s);
1008
1009            par.X = Wh * Xm;
1010            par.M = m;
1011            par.N = s;
```

```
1012
1013            vec info;
1014            vec Wt = cvectorize(eye(m));
1015            vec opts = "1 1e-4 1e-8";
1016            opts << maxit;
1017
1018            ucminf(info, par, Wt, opts);
1019
1020            w = (reshape(Wt,m,m)*Wh)/scaleX;
1021            S = w * Xr;
1022            set_ncomp(tn);
1023    }
1024
1025    /* Modificar número de iterções da classe IcaML */
1026    void IcaML::set_maxiterations(const int it)
1027    {
1028            this->maxit = it;
1029    }
1030
1031    /* Retorno do número de iterações da classe IcaML */
1032    int IcaML::get_maxiterations(void) const
1033    {
1034            return maxit;
1035    }
1036    /* Reset do número de iterações da classe IcaML */
1037    void IcaML::reset_icaml(void)
1038    {
1039            this-> maxit = 2000;
1040    }
1041
1042    /* Destrutor da classe IcaML */
1043    IcaML::~IcaML()
1044    {
1045
1046    }
1047
1048    //-------------------------------------------------------------//
1049
1050    /* Cálculo rápido da operação: soma de (vector x matriz transposta) com
            (vector transposto x matriz) */
1051    static void sum_w_hT_h_wT(mat &m, vec &v1, vec &v2)
1052    {
1053            const double *v1p, *v2p, *v1pp, *v2pp;
1054            int i,j;
1055
1056            v1p = v1._data();
1057            v2pp = v2._data();
1058
1059            for (i=0;i<m.rows();i++)
1060            {
1061                    v2p = v2._data();
1062                    v1pp = v1._data();
1063                    for (j=0;j<m.cols();j++)
1064                    {
```

```
1065                            m._elem(i,j) += (*v1p * *(v2p++)) + (*v2pp * *(
                                    v1pp++));
1066                    }
1067                    v1p++;
1068                    v2pp++;
1069            }
1070    }
1071
1072    /* Soma dos valores das colunas de uma matriz retornando um vector */
1073    static vec sumcol(const mat A)
1074    {
1075            vec out = zeros(A.cols());
1076
1077            for ( int i= 0; i< A.cols(); i++ )
1078            {
1079                    out(i) = sum(A.get_col(i));
1080            }
1081
1082            return out;
1083    }
1084
1085    /* Ortogonalização de uma matriz */
1086    static mat orth(const mat M)
1087    {
1088            mat Q;
1089            mat U, V;
1090            vec S;
1091            double eps = 2.2e-16;
1092            double tol= 0.0;
1093            int mmax= 0;
1094            int r= 0;
1095
1096            svd( M, U, S, V );
1097
1098            if ( M.rows() > M.cols() )
1099            {
1100                    U = U( 0, U.rows()-1, 0, M.cols()-1 );
1101                    S = S( 0, M.cols()-1 );
1102            }
1103
1104            mmax = (M.rows()>M.cols())?M.rows():M.cols();
1105
1106            tol = mmax*eps*max( S );
1107
1108            for ( int i= 0; i< size(S); i++ )
1109            {
1110                    if ( S(i) > tol ) r++;
1111            }
1112
1113            Q = U( 0,U.rows()-1, 0, r-1 );
1114
1115            return (Q);
1116    }
1117
```

```
1118 /* Cálculo do expoente de uma matriz */
1119 static mat mpower(const mat A, const double y)
1120 {
1121         mat T = zeros( A.rows(), A.cols() );
1122         mat dd = zeros( A.rows(), A.cols() );
1123         vec d = zeros( A.rows() );
1124         vec dOut = zeros( A.rows() );
1125
1126         eig_sym( A, d, T );
1127
1128         dOut = pow( d, y );
1129
1130         diag( dOut, dd );
1131
1132         for ( int i= 0; i< T.cols(); i++ )
1133         {
1134                 T.set_col(i, T.get_col(i)/norm(T.get_col(i)));
1135         }
1136
1137         return ( T*dd*transpose(T) );
1138 }
1139
1140 /* Operador especial para adicionamento de valores a um vector */
1141 static vec &operator << (vec &os, const double value)
1142 {
1143         os.ins(os.size(),value);
1144         return os;
1145 }
1146
1147 /* Operador especial para adicionamento de um vector a um vector */
1148 static vec &operator << (vec &os, const vec v1)
1149 {
1150         os.ins(os.size(),v1);
1151         return os;
1152 }
1153
1154 /* Armazenamento de uma matriz para um ficheiro através de um operador
       especial */
1155 static fstream &operator << (fstream &file, const mat &m)
1156 {
1157         if (file.is_open())
1158         {
1159                 string d = to_str(m);
1160
1161                 for(int i=0; i<d.size();i++)
1162                 {
1163                         if ((d[i]=='[')||(d[i]==']'))
1164                         {
1165                                 d.erase(i,1);
1166                                 i--;
1167                         }
1168                 }
1169
1170                 file.write(d.c_str(),d.size());
```

```
1171            }
1172
1173
1174            return file;
1175    }
1176
1177
1178    static vec diff(vec &v1)
1179    {
1180            int m = v1.size();
1181            int n = m;
1182
1183            vec v2;
1184
1185            while (n != 1)
1186            {
1187                    v2 << (v1(m-n+1)-v1(m-n));
1188                    n--;
1189            }
1190
1191            return v2;
1192    }
1193
1194    static vec diff(vec v1)
1195    {
1196            int m = v1.size();
1197            int n = m;
1198
1199            vec v2;
1200
1201            while (n != 1)
1202            {
1203                    v2 << (v1(m-n+1)-v1(m-n));
1204                    n--;
1205            }
1206
1207            return v2;
1208    }
1209
1210    /* Cálculo da norma de um vector */
1211    static double norm_inf(const vec &v1)
1212    {
1213            return max(abs(v1));
1214    }
1215
1216    /* Cáculo da matriz de separação para o algoritmo IcaML (Infomax)*/
1217    static void ica_mlf(double &f, vec &dW, const param &par, const vec &W)
1218    {
1219
1220            mat Wt = reshape(W, par.M, par.M);
1221
1222            mat S = Wt * par.X;
1223
```

```
1224            f=-( par.N*log(abs(det(Wt))) - sum(sum(log( cosh(S) ))) - par.N
                    *par.M*log(pi) );
1225
1226            dW = cvectorize(-(par.N*inv(transpose(Wt)) - tanh(S)*transpose(
                    par.X)));
1227
1228    }
1229
1230    /* Cáculo da interpolação para a função softline*/
1231    static double interpolate(mat &xfd, int &n)
1232    {
1233            double eps = 2.2204e-16;
1234            double a = xfd(0,0);
1235            double b = xfd(1,0);
1236            double d = b - a;
1237            double dfia = xfd(0,2);
1238            vec C = diff(cvectorize(xfd(0,1,1,1))) - d*dfia;
1239
1240            if (C(0) >= 5*n*eps*b)
1241            {
1242                    double A = a - (.5*dfia*(pow(d,2)/C(0)));
1243                    d = 0.1*d;
1244                    return (min(max(a+d, A), b-d));
1245            }
1246            else return ((a+b)/2);
1247    }
1248
1249    /* Função para aproximação aos valores de update para o algoritmo BFGS
        */
1250    static void softline(double &alpha, double &F, vec &g, int &neval,
        double &slrat, const param &par, const vec &x, const vec &h)
1251    {
1252
1253            alpha = 0; neval = 0;  slrat = 1; int n = x.size();
1254
1255            double dfi0 = dot(h,g);
1256
1257            if (dfi0 >= 0) return;
1258
1259            double fi0 = F; double slope0 = .05*dfi0; double slopethr =
                    .995*dfi0;
1260            double dfia = dfi0; double fib, dfib, c;
1261
1262            bool stop = 0; bool ok = 0;
1263
1264            neval = 0; double b = 1;
1265            vec update, gib;
1266            mat xfd(3,3);
1267
1268            while (!stop)
1269            {
1270                    update = x+b*h;
1271                    ica_mlf(fib, gib, par, update);
1272
```

```
1273                    neval++;
1274                    dfib = dot(gib,h);
1275
1276                    if  (b == 1)
1277                            slrat = dfib/dfi0;
1278
1279                    if  (fib <= fi0 + slope0*b)
1280                    {
1281                            if (dfib > abs(slopethr))
1282                                    stop = 1;
1283                            else
1284                            {
1285                                    alpha = b;
1286                                    dfia = dfib;
1287                                    ok = 1;
1288                                    F = fib;
1289                                    g = gib;
1290                                    slrat = dfib/dfi0;
1291
1292                                    if  ((neval < 5) && (b < 2) && (dfib <
                                            slopethr))
1293                                            b = 2*b;
1294
1295                                    else stop = 1;
1296
1297                            }
1298                    }
1299                    else stop = 1;
1300            }
1301
1302            stop = ok;
1303
1304            xfd(0,0) = alpha;
1305            xfd(0,1) = F;
1306            xfd(0,2) = dfia;
1307            xfd(1,0) = b;
1308            xfd(1,1) = fib;
1309            xfd(1,2) = dfib;
1310            xfd(2,0) = b;
1311            xfd(2,1) = fib;
1312            xfd(2,2) = dfib;
1313
1314            //...............//
1315            while (!stop)
1316            {
1317                    c = interpolate(xfd, n);
1318
1319                    update = x+c*h;
1320                    ica_mlf(fib, gib, par, update);
1321                    neval++;
1322
1323                    xfd(2,0) = c;
1324                    xfd(2,1) = fib;
1325                    xfd(2,2) = dot(gib,h);
```

```
1326
1327                    if (fib < (fi0 + slope0*c))
1328                    {
1329                            xfd.copy_row(0, 2);
1330                            ok = 1;
1331                            alpha = c;
1332                            F = fib;
1333                            g = gib;
1334                            slrat = xfd(2,2)/dfi0;
1335                    }
1336
1337                    else
1338                    {
1339                            xfd.copy_row(1, 2);
1340                            ok = 0;
1341                    }
1342
1343                    ok = ok && (abs(xfd(2,2)) <= abs(slopethr));
1344                    stop = ok || (neval >= 5) || ((diff(cvectorize(xfd
                           (0,1,0,0)))(0))<= 0);
1345            }
1346
1347            //................//
1348
1349    }
1350
1351    /* Verificação de dados para o algoritmo IcaML*/
1352    static bool check(double &F, vec &g, int &n, param &par, vec &x, vec &
           opts)
1353    {
1354            n = size(x);
1355            ica_mlf(F, g, par, x);
1356            if (g.size() != x.size())
1357            {
1358                    cout << "Derivada com tamanho incorrecto!" << endl;
1359                    return 0;
1360            }
1361
1362            if (size(opts) != 4)
1363            {
1364                    opts.set_size(0);
1365                    opts << 1 << (1e-4 * norm_inf(g)) << 1e-8 << 100;
1366            }
1367            return 1;
1368    }
1369
1370
1371    /* Algoritmo de optimização não-linear BFGS para convergência no
           cálculo da matriz de separação */
1372    static void ucminf(vec &info, param &par, vec &x, vec &opts)
1373    {
1374
1375            double F;
1376            vec g, vm;
```

```
1377        int n, fst;
1378
1379        if (!check(F, g, n, par, x, opts)) return;
1380
1381        mat D = eye(n);
1382        fst = 1;
1383
1384        int k = 0, kmax = (int) opts(3), neval = 1, dval;
1385
1386        double Delta = opts(0), ng = norm_inf(g);
1387        double nh = 0, eps = 2.2204e-16, Fp, nx, al, slrat, yh, yv, a,
               thrx;
1388        int found = ng <= opts(1);
1389
1390        bool red;
1391        vec xp, gp, y, v, w, h;
1392        mat th, tw, mt;
1393
1394        //..............Inicio do While.....................////
1395
1396        while (!found)
1397        {
1398                xp = x;
1399                gp = g;
1400                Fp = F;
1401                nx = norm(x);
1402
1403                h = D*(-gp);
1404
1405                nh = norm(h);
1406                red = 0;
1407
1408                if  (nh <= (opts(2)*(opts(2)+nx)))
1409                        found = 2;
1410                else
1411                {
1412                        if  ((Delta < fst) || (nh > Delta))
1413                        {
1414                                h *= (Delta / nh);
1415                                nh = Delta;
1416                                fst = 0;
1417                                red = 1;
1418                        }
1419                }
1420
1421                k++;
1422
1423                softline(al, F, g, dval, slrat, par, x, h);
1424
1425                if  (al < 1)
1426                        Delta *= .35;
1427                else if (red & (slrat > .7))
1428                        Delta *= 3;
1429
```

```
1430
1431                    neval += dval;
1432                    ng = norm_inf(g);
1433
1434                    x += al*h;
1435                    h = x - xp;
1436                    nh = norm(h);
1437
1438                    if (nh == 0)
1439                            found = 4;
1440                    else
1441                    {
1442                            y = g - gp;    yh = dot(y,h);
1443
1444                            if  (yh > (sqrt(eps) * nh * norm(y)))
1445                            {
1446                                    //  Update  D
1447                                    v = D*y;
1448                                    yv = dot(y,v);
1449
1450                                    a = (1 + yv/yh)/yh;
1451                                    w = (a/2)*h - v/yh;
1452
1453                                    sum_w_hT_h_wT(D,w,h);
1454
1455                            }
1456
1457                            thrx = (opts(2)*(opts(2) + norm(x)));
1458                            if (ng <= opts(1)) found = 1;
1459                            else if (nh <= thrx) found = 2;
1460                            else if (neval >= kmax) found = 3;
1461
1462                            else Delta = max(Delta, 2*thrx);
1463                    }
1464
1465            }
1466
1467            info << F << ng << nh << k << neval << found;
1468
1469            //..............Fim do While.......................//
1470    }
1471
1472    /* Cálculo da pseudo inversa de uma matriz */
1473    static mat pinv(mat W)
1474    {
1475            int m = W.rows();
1476            int n = W.cols();
1477            vec S, s;
1478            mat U,V,res;
1479            double eps = 2.2204e-16, tol;
1480            int r = 0, mf, nf;
1481
1482            if (n > m)
1483                    return transpose(pinv(W.T()));
```

```
1484        else
1485        {
1486                svd(W, U, S, V);
1487                if (m>n) U = U(0,-1,0,n-1);
1488        }
1489
1490        if (m > 1)
1491                s = S;
1492        else if (m == 1)
1493                s << S(1);
1494        else
1495                s = "0";
1496
1497        tol = max(m,n) * max(s) * eps;
1498
1499        for(int i=0; i<s.size();i++)
1500        {
1501                if (s(i) > tol)
1502                        r++;
1503        }
1504        if (r == 0)
1505                return zeros((W.T()).rows(), (W.T()).cols());
1506        else
1507        {
1508                mat Sm = diag(elem_div(ones(r),s(0,r-1)));
1509                return (V(0,-1,0,r-1)*Sm*transpose(U(0,-1,0,r-1)));
1510        }}
```

Anexo I: Listagem do Código Interface Swig para Módulo pyIca

```
1  /*Ficheiro de código pyica.i do interface para Python desenvolvido
2  por Michael Guerreiro*/
3
4  %module pyICA
5
6  /* Livrarias a serem incluidas: a toolbox ica em c++ e a toolbox para
7  fazer o interface para Numeric */
8
9  %{
10 #include "ica.h"
11 #include "Numeric/arrayobject.h"
12 %}
13
14 %init %{
15    import_array();
16 %}
17
18 /* Estrutura para o acesso aos dados de uma classe Mix */
19
20 %inline
21 %{
22 struct MixRow
23 {
24      Mix *g;
25      int row;
26
27      double __getitem__(int i)
28      {
29           if ((row > g->get_mixt()-1)||(i > g->get_samples()-1)||(row <
                0)||(i < 0))
30           {
31                std::cout << "Valores fora de alcance!" << endl;
32                return 0;
33           }
34
35           return g->matrix(row, i);
36      };
37 };
38 %}
39
40 /* Função para conversão de matrizes e vectores c++ para arrays numeric
       */
41
42 %{
43 PyObject *Convert2Numpy(void *data, int rows, int cols)
44 {
45      PyArrayObject *numpy;
```

```
46      int dims[2];
47
48      dims[0] = rows;
49      dims[1] = cols;
50      numpy =  (PyArrayObject *) PyArray_FromDims(2, dims, PyArray_DOUBLE
            );
51
52      memcpy(numpy->data, data, dims[0]*dims[1]*sizeof(double));
53
54      return (PyObject*) numpy;
55  };
56  %}
57
58  /* Controlo de entrada de arrays numeric */
59
60  %typemap(python, in) PyArrayObject *
61  {
62      $1 = (PyArrayObject *)$input;
63
64      if ( ($1->nd <= 1) )
65      {
66          PyErr_SetString(PyExc_TypeError, "Não é uma matrix");
67          return NULL;
68      }
69
70      if ( ($1->descr->type_num != PyArray_DOUBLE) )
71      {
72          PyErr_SetString(PyExc_TypeError, "Matrix não do tipo DOUBLE");
73          return NULL;
74      }
75  }
76
77  /* Controlo de saida de vectores e conversão para arrays numeric */
78
79  %typemap(python, out) vec
80  {
81          $result = Convert2Numpy($1._data(),1,$1.size());
82  }
83
84  /* Controlo de saida de matrizes e conversão para arrays numeric */
85
86  %typemap(python, out) mat
87  {
88          $result = Convert2Numpy($1._data(),$1.rows(),$1.cols());
89  }
90
91  %include "ica.h"
92
93  %inline %{
94  /* Sub-classe da classe Mix para permitir a entrada de arrays numeric
        */
95  class Data: public Mix
96  {
97          public:
```

```
98                      Data() {
99                              reset();
100                     }
101
102                     Data(const Data &other) {
103                             matrix = other.matrix;
104                             m = other.get_mixt();
105                             s = other.get_samples();
106                             filename = other.get_filename();
107                     }
108
109                     Data(const Mix &other) {
110                             matrix = other.matrix;
111                             m = other.get_mixt();
112                             s = other.get_samples();
113                             filename = other.get_filename();
114                     }
115
116                     Data(char *lfilename, const int mixt, const int samples
                            ) {
117                             load_matrix(lfilename, mixt, samples);
118                     }
119
120                     Data(PyArrayObject *array) {
121                             set_matrix(mat((double *)array->data, array->
                                    dimensions[0], array->dimensions[1]));
122                     }
123
124                     ~Data(){
125                     }
126 };
127 %}
128 /* Extensão para o acesso aos dados de uma classe Mix */
129 %extend Mix {
130
131         MixRow __getitem__(int i)
132         {
133                 MixRow r;
134                 r.g = self;
135                 r.row = i;
136                 return r;
137         };
138 };
```

Anexo J: Script Python para busca da melhor posição inicial no cálculo da localização dipolar

```
1  #Ficheiro de código SearchPosIni.py. Script para a busca da posição
       inicial desenvolvido por Michael Guerreiro
2  #Este script utiliza o programa dipoli desenvolvido por Thom Oostendorp
3
4  import os, subprocess, string, time
5
6  guess = ["   1 \n", "\n", "   1 position stationary orientation free "]
7
8  def execute():
9      p = subprocess.Popen( "dipoli geo_matrix3.dat -m dadost1.dat -s
           first_guess.dat -o test", stdin = subprocess.PIPE, stdout =
           subprocess.PIPE, stderr = subprocess.PIPE)
10     stdout_text, stderr_text = p.communicate()
11     return stdout_text.rstrip()
12
13 def setInicialPoint(x, y, z):
14     guess[1] = "   1   " + `x` + "   " + `y` + "   " + `z` + "\n"
15     f = open('first_guess.dat', 'wb')
16     f.writelines(guess)
17     f.close()
18 def findRV(lstr):
19     for t in lstr:
20       if t[0:2]==" R":
21             return t
22 def findPos(lstr):
23     i = 0;
24     for t in lstr:
25       if t[0:2]==" R":
26             return lstr[i-1]
27       i=i+1
28 def findOri(lstr):
29     i = 0;
30     for t in lstr:
31       if t[0:2]==" R":
32
33             return lstr[i-4].lstrip().rstrip()+lstr[i-3].rstrip()+lstr[i
                   -2].rstrip()
34       i=i+1
35
36 x=[0]
37 y=[-7]
38 z=[0]
39
40 for i in range(1,21):
41   x.append(x[i-1]+1)
```

```
42    z.append(z[i-1]+1)
43
44  for i in range(1,13):
45    y.append(y[i-1]+1)
46
47  rvs = []
48  lstrpos = []
49  posini = []
50  posfim = []
51  orient = []
52  rvl = []
53
54  for xx in x:
55    for yy in y:
56      for zz in z:
57
58        setInicialPoint(xx, yy, zz)
59
60        str1 = execute()
61        lstr1 = str1.split("\n")
62
63        srv = findRV(lstr1)
64        pos = findPos(lstr1)
65        ori = findOri(lstr1)
66
67        if srv:
68          lpos = pos.split(",")
69          lstrpos.append("point␣" + lpos[0].split("(")[1].lstrip() + "␣"
              + lpos[1].lstrip("␣") + "␣" + lpos[2].lstrip("␣").rstrip().
              rstrip(")") + "\n")
70          lrv = srv.split("␣")
71          rv = string.atof(lrv[-1].rstrip())
72          rvl.append(lrv[-1].rstrip())
73
74          orient.append(ori)
75          posini.append(‘xx‘ + "␣" + ‘yy‘ + "␣" + ‘zz‘)
76          posfim.append(lpos[0].split("(")[1].lstrip() + ’␣’ + lpos[1].
              lstrip("␣") + ’␣’ + lpos[2].lstrip("␣").rstrip().rstrip(")")
              )
77          rvs.append(rv)
78
79  tot = []
80  i=0
81  for x in rvl:
82          tot.append(x + ’␣␣␣␣␣’ + posini[i] + ’␣␣␣␣␣’ + posfim[i] + ’␣␣␣
              ␣␣’ + orient[i] + ’\n’)
83          i=i+1
84
85  tot.sort()
86  f = open(’c:\point’, ’wb’)
87  f.writelines(lstrpos)
88  f.close()
```