UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

DANILO MENDES DIAS DELFINO DA SILVA

Desenvolvimento de console multiplataforma para aquisição,
organização e visualização de dados do espectrômetro digital de RM
do CIERMag: ToRM Console

São Carlos
2014

DANILO MENDES DIAS DELFINO DA SILVA

Desenvolvimento de console multiplataforma para aquisição,
organização e visualização de dados do espectrômetro digital de RM
do CIERMag: ToRM Console

Dissertação apresentada ao Programa de
Pós-Graduação em Física do Instituto de
Física de São Carlos da Universidade de
São Paulo, para obtenção do título de
Mestre em Ciências.

Área de concentração: Física Aplicada
Opção: Física Computacional
Orientador: Prof. Dr. Alberto Tannús

Versão Corrigida

(Versão original disponível na Unidade que aloja o Programa)

São Carlos
2014

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Silva, D. M. D. D.

Desenvolvimento de console multiplataforma para aquisição, organização e visualização de dados do espectrômetro digital de RM do CIERMag: ToRM Console / D. M. D. D. Silva; orientador Alberto Tannús - versão corrigida -- São Carlos, 2014.
143 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Física Aplicada Computacional) -- Instituto de Física de São Carlos, Universidade de São Paulo, 2014.

1. Automação de RM. 2. Desenvolvimento de Software. 3. Relaxometria. I. Tannús, Alberto, orient. II. Título.

À minha mãe, Patricia, aos meus avós, Seu Lázinho e Dona Cida.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Alberto Tannús, pela amizade, pela confiança, pelos ensinamentos técnicos passados através de longas e proveitosas discussões e pela oportunidade que me foi conferida de trabalhar em um grupo tão produtivo e proativo.

Ao Dr. Mateus J. Martins e ao Dr. Edson L. G. Vidoto, pela amizade, pelo apoio técnico e pelas valiosas discussões acerca das inúmeras dificuldades encontradas durante este trabalho.

Aos demais colegas do CIERMag, pela amizade e contribuição direta na realização deste trabalho através de suas responsabilidades no grupo (Prof. Dr. Fernando Paiva, Mário, Guilherme, Gustavo, Pizetta, Felipe, Tamires, Cíntia, Faimison, Erick, Pedro, Jackeline, Roberson, Carlinhos, Ricardo, Jorge, Mariana, Gelia, Heitor, Daniel Consalter, Nilzeli e Patrícia).

À minha mãe, Patricia, pelo amor e carinho e pelos incansáveis esforços sem os quais eu não teria condições de ter alcançado os conhecimentos necessários para realização deste trabalho.

Aos meus avós, Lázaro e Aparecida, e à tia Wilma pelo carinho e compreensão pela distância e saudade.

Aos moradores (Cabeça, Pudim, Yole, Manguaço e Arroz), ex-moradores (Zé, Baiano, Tucano, Thomaz e Digon) e agregados (Cesão, Pelé, David, B2, Bessa, Cibele, Laís e Fuca) da República das Bermudas, que, entre risos, copos e outros, proporcionaram momentos de descontração, felicidade, reflexão e incentivo durante a realização deste trabalho.

A todos os amigos barretenses pela amizade de longa data que, apesar da distância, foi mantida e fortalecida ainda mais neste período. Em especial, aos amigos Ganso, Haroldo, Flávia e Murilão por terem proporcionado as mais reflexivas conversas de bar.

Ao IFSC-USP, pelo programa de pós-graduação e infraestrutura às agências de fomento pelo incentivo financeiro ao projeto (FAPESP: 2005/5.6663-1 e CNPQ: 565.047/2010-8, FINEP, FNS) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior pela concessão da bolsa de mestrado.

“Ninguém além de nós mesmos pode libertar nossa mente.”

Bob Marley

RESUMO

SILVA, D. M. D. D. **Desenvolvimento de console multiplataforma para aquisição, organização e visualização de dados do espectrômetro digital de RM do CIERMag: ToRM Console**. 2014. 143 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.

As técnicas de ressonância magnética nuclear (RM), por não serem invasivas ou destrutivas, apresentam um papel importante em diversas áreas do conhecimento como, por exemplo, medicina, química, física, setor rural, setor petrolífero, *etc*. Entretanto os custos de aquisição, manutenção e atualização dos equipamentos comerciais são muito altos e países em desenvolvimento, como é o caso do Brasil, sofrem com os impactos em função deste fator, principalmente nos setores ligados à pesquisa, em especial, nas áreas de concepção de novos métodos de RM. Além disso, o cenário nacional de desenvolvimento tecnológico desta área é escasso e os equipamentos comerciais fornecidos pelas empresas nem sempre oferecem a flexibilidade desejada. Baseado no uso de tecnologias inovadoras, como é o caso dos *hardwares* reconfiguráveis e uso de linguagem, compilador, bibliotecas e editores próprios para desenvolvimento de métodos de sequência de pulsos de RM, o Centro de Imagens e Espectroscopia *in vivo* por Ressonância Magnética (CIERMag) propõe uma solução alternativa através do desenvolvimento de um espectrômetro completamente digital de RM, flexível o suficiente para operar como um tomógrafo. O objetivo deste trabalho é idealizar e desenvolver a camada de *software* responsável pela aquisição, organização, visualização e armazenamento dos dados provenientes do espectrômetro através do subsistema *ToRM Console*, levando em consideração a flexibilidade proposta. Além disso, um arcabouço de ferramentas para automatização de pós-processamento, o *ToRM Processing API*, foi desenvolvido. Ambos os sistemas foram implementados em linguagem *Python*, com auxílio das principais bibliotecas de propósito científico (*Numpy*, *Scipy*, *Matplotlib*, *etc*). A realização de um experimento para estudo do comportamento do tempo de relaxação transversal de soluções de sulfato de cobre em função da variação da concentração é mostrada através do uso do método Carr-Purcell-Meiboom-Gill (CPMG). Outras aplicações do espectrômetro são mostradas através da obtenção de dados gerados por meio de sintetizadores de sinais acoplados ao espectrômetro. Os resultados mostram que o espectrômetro atualmente pode operar como um relaxômetro de RM de fácil operação. Os sistemas desenvolvidos neste

trabalho mostram-se suficientemente flexíveis e preparados para operação em experimentos de Imagens e Espectroscopia de Alta Resolução, que são futuras etapas de desenvolvimento no CIERMag, tendo em vista as ferramentas desenvolvidas e testadas através do uso de dados sintetizados.

Palavras-chave: Automação de RM. Desenvolvimento de *software*. Relaxometria.

ABSTRACT

SILVA, D. M. D. D. **Development of a multiplatform console for acquisition, organization and visualization of CIERMag digital spectrometer data:** ToRM Console. 2014. 143 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2014.

The Nuclear Magnetic Resonance (NMR) techniques, which are neither invasive nor destructive, play an important role on several areas of knowledge, such as Medicine, Chemistry, Physics, Agriculture, Oil industry, among others. However, the costs associated with acquisition, maintenance and upgrade are very high, which have impacted on developing countries, as Brazil. Furthermore, the national scenario of technological development in this area is scarce and the commercial equipment provided by the companies does not always offer the desired flexibility. Based on the use of innovative technologies, which is the case of reconfigurable hardware and the use of proprietary language, compiler, libraries and editors for the development of NMR sequence pulses methods, the Magnetic Resonance Imaging and *in vivo* Spectroscopy Center (CIERMag) have proposed an alternative solution through the development of a completely digital NMR spectrometer, flexible enough to operate as an MRI Scanner. This dissertation reports the design and development of a software layer responsible for the acquisition, organization, visualization and storage of the data provided by the spectrometer through the ToRM Console subsystem, taking into account the proposed flexibility. A framework for post-processing automation, the ToRM Processing API, is also described. Both systems were developed in Python programming language, with the help of the main scientific purpose libraries (Numpy, Scipy, Matplotlib, *etc.*). Carr-Purcell-Meiboom-Gill (CPMG) method was used for the evaluation of the transverse relaxation time dependency on variable concentrations of copper sulfate solution. Other spectrometer applications are shown through the acquisition of synthesized data from simulation devices coupled to the spectrometer. The results show that the spectrometer can currently operate as an easy-to-use NMR Relaxometer. The systems described in the present work proved to be sufficiently flexible and prepared for the operation as MRI Scanners or Analytical Spectrometers, which are the future stages of development at CIERMag, considering the tools developed and tested through the use of synthesized data.

Keywords: NMR automation. Software development. Relaxometry.

LISTA DE FIGURAS

- Figura 1 - Valores possíveis para a componente L_z do *momentum* angular para o caso $l = 1$ 38
- Figura 2 - comportamento do momento magnético μ sujeito a um campo magnético B , conforme a equação (16). 41
- Figura 3 - Evolução temporal da magnetização, levando em consideração os mecanismos de relaxação. Inicialmente, toda magnetização encontra-se no plano transversal. Esta situação pode ser obtida com a aplicação de um pulso de radiofrequência, que será vista na subseção 2.2.5. Em seguida, ela retorna a sua condição de equilíbrio, conforme previsto pela equação de Bloch..... 45
- Figura 4 - Esquema simplificado de uma bobina transmissora. O campo produzido pela bobina, $B_1(t)$, aponta sempre ortogonalmente ao campo estático B_0 . Além disso, a frequência de oscilação deste campo deve ser igual à frequência de Larmor. 50
- Figura 5 - O campo oscilatório de intensidade $B_1(t) = 2B_1\cos(\omega_0 t)$ pode ser decomposto em duas componentes, $B_1 +$ e $B_1 -$. Quando observadas em um *referencial de coordenadas girantes*, que gira em torno do eixo z com frequência de Larmor, apenas uma dessas componentes é efetivamente observada, levando em consideração somente os efeitos de primeira ordem..... 50
- Figura 6 - Após a aplicação de um pulso com duração τ , a magnetização é deslocada de um ângulo $\theta = \gamma B \tau$ em torno do campo magnético B que, neste caso, aponta para a direção x (esquerda). Quando a magnetização é deslocada de um ângulo de 90 graus, utiliza-se o termo ‘aplicação de um pulso de $\pi/2$ ’ (centro). De maneira análoga, quando a magnetização sofre uma inversão no seu sentido, é utilizado um ‘pulso de π ’ (direita). 51
- Figura 7 - Decaimento livre da indução (FID) em azul. A envoltória vermelha representa a evolução do módulo da magnetização transversal. O tempo de relaxação T_2^* é utilizado em função da inomogeneidade de campo presente nos experimentos reais..... 52
- Figura 8 - Evolução temporal da magnetização durante a execução de um método *spin-eco*. Inicialmente a magnetização está no seu estado de equilíbrio (A). Em seguida, um pulso de $\pi/2$ na direção $-x$ é aplicado, de maneira que a magnetização resultante aponte para direção y (B). O processo de relaxação transversal começa a atuar em suas duas formas simultaneamente: por meio de efeitos termodinâmicos; e por meio de inomogeneidade de campo. A última faz com que os momentos magnéticos de cada um dos núcleos, representados fora de escala por ‘a’, ‘b’, ‘c’, ‘d’ e ‘e’, precessionem com frequências de Larmor diferentes, causando perda de coerência (C). Após

um tempo TE , a aplicação de um pulso de π na direção y faz com que as fases sejam invertidas, de modo que os momentos magnéticos comecem a precessionar com velocidade angular oposta a da situação anterior (D). No tempo $2TE$, os momentos magnéticos entram novamente em coerência, maximizando o valor da magnetização. Nesta situação o decaimento do valor da magnetização deve-se somente aos efeitos termodinâmicos e, portanto, a leitura do sinal neste momento pode fornecer o valor do tempo de relaxação T_2 (E). Por fim, se não houver a aplicação de nenhum outro pulso, a magnetização transversal e longitudinal retornam aos seus respectivos valores de equilíbrio (F).54

Figura 9 - Espectrômetro digital do CIERMag ligado ao osciloscópio para realização de testes.56

Figura 10 - Estrutura do código de um método de sequência de pulsos escrito na linguagem F. Um mesmo método possui um único trecho de código denominado *sequence* (sequência) que pode conter um ou mais *modules* (módulos) que, por sua vez, podem conter um mais *events* (eventos). Módulos distintos podem fazer referência aos mesmos eventos, conforme indicado na figura. Além disso, dentro dos módulos estão contidas as definições dos *loops* que determinam a sequência de pulsos.59

Figura 11 - Trecho do módulo '*Balanced*' escrito em linguagem F. A inicialização do módulo ocorre entre as linhas 29 e 35, onde o mesmo recebe os parâmetros oriundos do protocolo de execução. As linhas 37, 42, 43, 44, 46, 49 e 51 são chamadas para eventos, que estão definidos em outros arquivos que devem ser importados no início do código. As linhas 38, 39 e 40 definem *loops*, ou iterações, de maneira que o primeiro parâmetro define o nome da variável atribuída ao índice do *loop* e o valor inicial do mesmo, o segundo parâmetro define o último valor do índice e o parâmetro *type* define o tipo de loop, que será discutido no capítulo 4. Os *loops* são finalizados nas linhas 52, 53 e 54. As linhas 45 e 47 são *delays*, ou atrasos, utilizados para que o espectrômetro aguarde um determinado intervalo de tempo até a execução da próxima instrução. A linha 55 indica o término do módulo '*Balanced*'.60

Figura 12 - ToRM IDE em execução no gerenciamento de parâmetros do método '*CPMG_E*'.62

Figura 13 - Eclipse com o *plugin* do editor F em execução durante a edição de uma biblioteca de eventos.63

Figura 14: Editor gráfico de sequências sendo executado dentro do escopo do *ToRM IDE*. O módulo '*SE_Multislice*' é mostrado.63

Figura 15 - *ACQServer* em funcionamento durante a carga do método '*CPMG_E*'. Os comandos recebidos foram enviados pelo *ToRM Console*, conforme será discutido no capítulo 4.64

Figura 16 - Hierarquia do diretório do sistema. Nesta figura estão contidos apenas os subdiretórios que o compõe, omitindo-se os arquivos.68

Figura 17 - <i>CIERMag Workplace</i> . Este diretório, semelhante ao diretório raiz do sistema, é de uso particular de cada usuário.	70
Figura 18 - Tela inicial (<i>System Login</i>) do <i>ToRM Console</i> . Os usuários pré-configurados no arquivo ' <i>usr_cfg.json</i> ' são listados no campo ' <i>User</i> ' e a respectiva senha deve ser digitada no campo ' <i>Password</i> '. Os campos ' <i>Database Host</i> ' e ' <i>ACQServer Host</i> ' informam, respectivamente, o endereço da base de dados referente ao sistema de usuários e o endereço onde o servidor de aquisições, <i>ACQServer</i> , está localizado.	71
Figura 19 - Diagrama de relacionamento das tabelas que constituem o sistema de usuários.	71
Figura 20 - Gerenciamento de conta. Visualização e alteração de dados pessoais.	72
Figura 21 - Gerenciamento de contas. Administração de contas do sistema.	73
Figura 22 - Gerenciamento de conta. Criação de novos usuários.	73
Figura 23 - Gerenciamento de permissões do <i>ToRM Console</i> . Nesta janela é possível controlar as permissões disponíveis para cada um dos grupos, caso o usuário autenticado possua permissão para tal tarefa.	74
Figura 24 - Tela de requisição de permissão temporária do <i>ToRM Console</i> . É possível que um usuário pertencente a um grupo de hierarquia maior possa conceder permissão temporária ao operador corrente.	75
Figura 25 - Tela inicial do <i>ToRM Console</i> após etapa de autenticação. Na árvore de itens ' <i>Methodologies</i> ' estão contidas todos os métodos disponíveis ao usuário. Os métodos que estão sob o item ' <i>System Methodologies</i> ' estão disponíveis a todos os usuários, enquanto as que estão sob o item, neste caso, ' <i>danilo</i> ' são pertencentes a este usuário.	76
Figura 26 - <i>ToRM Console</i> após carregamento do método <i>SE_Mul</i> . O sistema de visualização, ' <i>Viewport</i> ', fica em espera e a aba de parâmetros permite a alteração dos valores dos parâmetros.	78
Figura 27 - Interação entre os subsistemas do espectrômetro do <i>CIERMag</i> , levando em consideração a arquitetura cliente-servidor utilizada. A edição, compilação e o controle e preparação do <i>scan</i> são feitos no lado do cliente, enquanto que a comunicação direta com o <i>hardware</i> é feita através do servidor. O <i>ToRM Console</i> (cliente) e o <i>ACQServer</i> (servidor) constituem a ponte de comunicação entre os dois lados.	81
Figura 28 - Sistema de aquisição do <i>ToRM Console</i> . Os dados são adquiridos do <i>ACQServer</i> pelo <i>SpecClient</i> através do protocolo FTP e enviados para o processo de filtragem. Concorrentemente, os dados filtrados são enviados ao sistema de visualização, que é executado no processo principal. Além disso, os dados são armazenados em memória no processo principal e gravados em disco no final da execução do experimento. Todas estas etapas são gerenciadas pela classe <i>ExperimentController</i>	84

- Figura 29 - Sistema de visualização do *ToRM Console (Viewport)* no domínio do tempo. É possível observar a parte real (azul) e imaginária (verde) e o módulo (vermelho) do sinal. No canto inferior direito está contido o *IndexWidget*, informando o índice do dado visualizado. No menu superior esquerdo estão, da esquerda para direita os botões de *zoom* vertical e horizontal, o botão de ajuste de escala no dado atual, o botão de auto escala, o botão de chaveamento entre domínio do tempo e frequência e o botão de chaveamento entre visualização do último dado ou da média entre eles. No menu inferior à direita estão os botões de deslocamento dos eixos com auxílio do ponteiro do *mouse*, a ferramenta de *zoom*, com a definição da região também através do *mouse*, o botão de configuração de ajuste do tamanho da área de visualização e, por último, o botão de exportação de figuras.....86
- Figura 30 - Sistema de visualização do *ToRM Console (Viewport)* no domínio da frequência. O cálculo da Transformada de Fourier é realizado através do pacote *numpy.fft*. O processamento é realizado apenas para visualização, não afetando os dados que serão armazenados em disco.....87
- Figura 31 - Estrutura de *loops* ('*iterations*') contida no protocolo de aquisição do único módulo contido no método '*CPMG_E*'. Os valores dos índices '*Size*' de cada *loop* podem ser alterados por meio de parâmetros acessíveis à aba de parâmetros. Os demais valores são definidos no código F da sequência.....89
- Figura 32 - Evolução dos índices quando existem apenas os *loops* '*LoopExterno*' com 3 iterações (vermelho) e '*LoopInterno*' com 5 iterações (azul). O índice unidimensional (preto) varia de 0 a 14.....90
- Figura 33 - Exemplo adaptado da figura 32 para o caso em que o '*LoopInterno*' possui tipo '*Average*'. O índice referente ao '*LoopInterno*' é simplesmente removido.91
- Figura 34 - Geração de índices quando três *loops* são utilizados com tipos '*Split*' ('*LoopExterno*', em vermelho), '*Slice*' ('*LoopIntermediário*', em verde) e ordinário ('*LoopInterno*', em azul), em ordem decrescente de hierarquia. Todos os *loops* possuem duas iterações. As duas colunas representam a execução do primeiro (à esquerda) e segundo (à direita) '*Split*', embora sua dimensão seja truncada. O número efetivo de *slices* será, portanto, quatro.....92
- Figura 35 - Formato da função $bt=2fcsinc2fct$ no intervalo $[-3, 3]$ para o caso $fc = 1$ utilizando 1000 pontos.96
- Figura 36 - Problema observado após a aplicação de filtro FIR por meio de convolução com janela '*Hanning*'. O sinal (azul) representa a função $sen(\omega t)$, com $\omega = 1$ no intervalo $[0, 2\pi]$. A segunda curva representa a soma do sinal com um ruído gaussiano (vermelho). Embora o ruído de alta frequência tenha sido removido com a aplicação de um filtro FIR com frequência de corte $\omega_c = 1.1\omega$, o sinal filtrado (verde) apresenta problemas relacionados aos efeitos de borda da filtragem. O número de coeficientes do filtro foi de $M = 255$ e o modo "*same*" do módulo *numpy.convolve* foi utilizado para remoção de $M - 1$ pontos inválidos pelo processo de convolução.98

Figura 37 - O mesmo problema observado na figura 36, porém utilizando a função $\exp(-t)$ no intervalo $[0, 5]$. O ruído de alta frequência é novamente removido, porém os efeitos transientes do filtro são encontrados na extremidade esquerda da curva do sinal filtrado. O mesmo número de coeficientes foi utilizado (255) e a filtragem também foi utilizada no modo “same”	98
Figura 38 - Solução para o problema relacionado aos efeitos de bordas da filtragem. O sinal original (azul) representa a função $\sin(\omega t)$, com $\omega = 1$ no intervalo $[0, 2\pi]$ com a adição de ruído gaussiano. Em seguida, são concatenados ao sinal pontos com os mesmos valores das bordas (vermelho). O processo de filtragem é realizado aparentando efeitos de bordas fora da região realmente pertencente ao sinal (verde). Por fim, a remoção dos pontos excedentes é realizada (marrom).....	99
Figura 39 - Solução aplicada a um decaimento exponencial ruidoso. O procedimento é análogo ao apresentado na figura 38.	99
Figura 40 - Visualização de conjunto de dados obtidos através do <i>ToRM Console</i> pelo <i>software</i> HDFView. É possível observar os valores numéricos, bem como obter visualização gráfica.	101
Figura 41 - Diversos estágios nos quais os dados adquiridos passam até a obtenção das informações resultantes. Inicialmente os dados são adquiridos pelo <i>ACQServer</i> e enviados ao <i>ToRM Console</i> (1), onde são filtrados (2), organizados (3) e armazenados em disco junto aos protocolos de aquisição e execução (4). Por fim, a execução do <i>script</i> de pós-processamento realiza as operações necessárias para obtenção da informação resultante, podendo ser uma imagem, uma curva, um número ou qualquer outro tipo de saída em função da aplicação (5).....	104
Figura 42 - estrutura do diretório onde se encontram os arquivos produzidos pelo sistema de aquisição do <i>ToRM Console</i>	107
Figura 43 - <i>ConsoleWidget</i> sendo executado de dentro do <i>InteractiveWidget</i> . O usuário pode inserir comandos que obedecem à sintaxe do <i>Python</i>	115
Figura 44 - Código <i>Python</i> necessário para visualização de uma imagem randômica de 256 x 256 pixels utilizando os recursos da <i>ToRM Processing API</i> , <i>PyQtGraph</i> e <i>Numpy</i> . A primeira etapa é a importação de todas as bibliotecas utilizadas (1). Em seguida, uma instância da classe <i>WidgetManager</i> é criada e seu respectivo método para inicialização de uma aplicação <i>PySide</i> é executado (2). Para que a imagem randômica seja criada, é feito uso da função <i>randn</i> , contida no pacote <i>numpy.random</i> , que retorna um objeto do tipo <i>Numpy Array</i> contendo um conjunto de pontos flutuantes, gerados a partir de uma distribuição normal, que será visualizado através do método <i>setImage</i> da instância da classe <i>ImageView</i> da biblioteca <i>PyQtGraph</i> (3). A janela interativa é inicializada e o espaço de variáveis que poderá ser utilizado pelo usuário é informado através do dicionário nomeado <i>namespace</i> . Além disso, o objeto <i>wm</i> , que representa o elemento gráfico contendo a imagem, é adicionado à janela interativa (4). Por fim, a	

aplicação do <i>PySide</i> é executada através do método <i>end_qtapp</i> da classe <i>WidgetManager</i> , permitindo com que o usuário possa interagir com a interface gráfica (5).	116
Figura 45 - Resultado do código executado na figura 44. A classe <i>ImageView</i> da biblioteca <i>PyQtGraph</i> permite a visualização da imagem contendo ferramentas básicas como, por exemplo, <i>zoom</i> (através da rolagem do <i>mouse</i>), controle de intensidade máxima e mínima.	117
Figura 46 - Utilização das classes <i>ProtocolManager</i> , <i>DataManager</i> e <i>PathManager</i> . Um conjunto de dados sintetizados deve ser colocado dentro do endereço “data/SE_Mul”, relativo ao <i>script</i> executado, que deve conter a mesma estrutura mencionada na seção 5.3. A classe <i>PathManager</i> é utilizada para se obter os caminhos dos protocolos de execução e aquisição e dos dados. Estas informações são utilizadas pelos métodos <i>open_protocols</i> da classe <i>ProtocolManager</i> e <i>read_data</i> da classe <i>DataManager</i> .	117
Figura 47 - Demonstração da leitura de dados e protocolos através da biblioteca <i>ToRM Processing API</i> . Os dados visualizados foram sintetizados com a mesma técnica que será mencionada no capítulo 6. São apresentados no gráfico a parte real (azul), imaginária (verde) e módulo (vermelho) de parte destes dados. Os protocolos podem ser acessados pela linha de comando por meio do objeto <i>pm</i> da classe <i>ProtocolManager</i> . Alguns métodos desta classe, já discutidos neste capítulo, são demonstrados.	118
Figura 48 - Comportamento de múltiplas janelas inseridas na janela interativa.	119
Figura 49 - Sistema em funcionamento no modo de testes através do uso de sintetizador de sinal. À esquerda o <i>ToRM Console</i> está sendo executado para aquisição do sinal. À direita o osciloscópio mostra parte das formas de ondas geradas pelo método em execução e abaixo dele o espectrômetro acoplado ao sintetizador é mostrado.	121
Figura 50 - Eco sintetizado com ruído em banda de 1 MHz. O conjunto amostral possui 8192 pontos coletados em fase e quadratura (representados em números complexos). A parte real (azul), imaginária (verde) e o módulo correspondente (vermelho) são apresentados simultaneamente.	122
Figura 51 - Conjunto de dados mostrado na figura 50 após a aplicação de filtro passa-baixa FIR com banda passante entre -31,250 kHz e 31,250 kHz. O ruído de alta frequência foi reduzido.	122
Figura 52 - Continuação do mesmo experimento mostrado na figura 51 e na figura 52 após a aplicação de 128 médias. O ruído presente na região espectral com módulo abaixo da frequência de corte também é eliminado.	123
Figura 53 - Método ‘SE_Mul’ em execução. A dimensão de médias é colapsada, resultando apenas duas dimensões no conjunto de dados armazenados.	123
Figura 54 - Método ‘Turbo_SE’ em execução. Apenas as dimensões relativas à codificação de fase e <i>slice</i> continuam após o procedimento de organização de dados.	124

Figura 55 - Execução de método 'SE_Mul' com a utilização de dados sintetizados via <i>software</i> através de imagem do logo do CIERMag. O dado visualizado representa uma linha no espaço k.	126
Figura 56 - <i>Script</i> de pós-processamento para método "SE_Mul' no mesmo experimento mostrando na figura 55. A partir da aplicação da transformada inversa de Fourier bidimensional sobre o conjunto de dados obtidos (espaço k, à esquerda) foi possível reconstruir a imagem (à direita).....	126
Figura 57 - Espectrômetro com magneto permanente de 0.5 T utilizado para a realização de testes de <i>hardware</i> e <i>software</i> do sistema. A foto à direita mostra o <i>probe</i> , onde é possível colocar amostras no interior de tubos de ensaio.	127
Figura 58 - <i>ToRM Console</i> executando o método 'CPMG_E' com a amostra D, que possui concentração de 1g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.	129
Figura 59 - <i>ToRM Console</i> executando o método 'CPMG_E' com a amostra C, que possui concentração de 0,54 g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.	129
Figura 60 - <i>ToRM Console</i> executando o método 'CPMG_E' com a amostra A, que possui concentração de 0,08 g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.	130
Figura 61 - <i>Script</i> de pós-processamento em execução com auxílio do <i>ToRM Processing API</i> para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência 'CPMG_E' com a amostra A. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.....	131
Figura 62 - <i>Script</i> de pós-processamento em execução com auxílio do <i>ToRM Processing API</i> para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência 'CPMG_E' com a amostra B. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.....	131
Figura 63 - <i>Script</i> de pós-processamento em execução com auxílio do <i>ToRM Processing API</i> para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência 'CPMG_E' com a amostra C. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.....	132
Figura 64 - <i>Script</i> de pós-processamento em execução com auxílio do <i>ToRM Processing API</i> para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência 'CPMG_E' com a amostra D. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.....	132
Figura 65 - Taxa de relação transversal, $R2 = 1/T2$, em função da concentração de CuSO_4 , apresentando comportamento linear, conforme previsto pela literatura.	133

LISTA DE TABELAS

Tabela 1 - Determinação do momento magnético nuclear em função da paridade do número de massa (M) e número atômico (Z).	39
Tabela 2 - Massa e concentração das amostras (A, B, C e D) de CuSO_4 utilizadas no experimento.	128
Tabela 3 - Valores de tempos de relaxação obtidos para cálculo da taxa de relaxação calculada a partir do valor de T_2 médio.	130

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
CIERMag	Centro de Imagens e Espectroscopia <i>in vivo</i> por Ressonância Magnética
CPMG	Carr-Purcell-Meiboom-Gill
DICOM	<i>Digital Imaging and Communications in Medicine</i>
FID	<i>Free Induction Decay</i>
FIFO	<i>First In, First Out</i>
FIR	<i>Finite Impulse Response</i>
FPGA	<i>Field-programmable gate array</i>
FTP	<i>File Transfer Protocol</i>
GUI	<i>Graphical User Interface</i>
GbE	<i>Gigabit Ethernet</i>
HTML	<i>HyperText Markup Language</i>
IRM	Imagem por Ressonância Magnética
ISI	<i>Institute for Scientific Information</i>
JSON	<i>JavaScript Object Notation</i>
MSPS	<i>Mega Samples Per Second</i>
PC	<i>Personal Computer</i>
RF	Radio-frequência
RM	Ressonância Magnética
RPE	Ressonância Paramagnética Eletrônica
USB	<i>Universal Serial Bus</i>
USPTO	<i>United States Patent and Trademark Office</i>
WWW	<i>World Wide Web</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	29
2	PRINCÍPIOS DE RESSONÂNCIA MAGNÉTICA.....	35
2.1	Introdução.....	35
2.2	Princípios físicos	36
2.2.1	Momento angular, <i>spin</i> e momento magnético.	36
2.2.2	Magnetização e comportamento de um próton sujeito a um campo magnético.....	40
2.2.3	Mecanismos de relaxação e equação de Bloch.....	43
2.2.4	Detecção de sinal	45
2.2.5	Aplicação de pulsos e condição de ressonância	49
2.2.6	Técnicas de aquisição: Decaimento livre da indução, T2* e <i>spin</i> -eco	52
3	ESPECTRÔMETRO DO CIERMAG.....	55
3.1	Introdução.....	55
3.2	Métodos de sequências de pulsos, Linguagem F e <i>ToRMC</i>	57
3.3	<i>ToRM IDE</i> , Editor gráfico de sequências e Editor F	61
3.4	ACQServer	64
4	TORM CONSOLE.....	65
4.1	Introdução.....	65
4.2	Interface Gráfica.....	66
4.3	Estrutura dos diretórios do sistema.....	67
4.4	Sistema de usuários	70
4.5	Preparação de <i>scan</i>	75
4.6	Validação e compilação.....	79

4.7 Sistema de aquisição	80
4.8 Protocolo de aquisição, organização de dados e tipos especiais de <i>loops</i>	87
4.9 Filtros	94
4.10 Armazenamento de dados	100
4.11 Documentação.....	101
5 TORM PROCESSING API.....	103
5.1 Introdução	103
5.2 Dependências, distribuição e instalação.....	105
5.3 Estrutura dos arquivos de saída de um experimento.....	106
5.4 <i>Managers</i>	108
5.4.1 <i>ProtocolManager</i>	108
5.4.2 <i>DataManager</i>	110
5.4.3 <i>PathManager</i>	111
5.4.4 <i>WidgetManager</i>	112
5.5 <i>Widgets</i>	113
5.5.1 <i>InteractiveWidget</i>	114
5.6 Exemplos de uso: utilização de protocolos e visualização de imagens e gráficos.....	115
6 RESULTADOS	121
6.1 Operação com dados sintetizados	121
6.2 Experimentos de relaxometria: CPMG para medidas de T2	127
7 CONCLUSÕES, LIMITAÇÕES E TRABALHOS FUTUROS	135
7.1 Conclusões	135
7.2 Limitações e trabalhos futuros	136
REFERÊNCIAS	139

1 INTRODUÇÃO

As técnicas de ressonância magnética nuclear (RM) apresentam um papel importante em diversas áreas do conhecimento como, por exemplo, na medicina, na química, na física, no setor rural, no setor petrolífero, *etc.* Parte dessa importância deve-se ao fato de que a aplicação da técnica de RM tem algumas características que são fundamentais em diversas aplicações, como, por exemplo, não ser invasiva, destrutiva e não utilizar radiação ionizante.

No entanto, estudos sobre a difusão e incorporação da tecnologia de ressonância magnética no Brasil demonstraram que a participação do país é consideravelmente baixa em relação aos países desenvolvidos. No período de 1976-2006, apenas 1% dos artigos publicados nas bases de dados do *Institute for Scientific Information* (ISI) são brasileiros e apenas uma patente brasileira foi registrada nas bases de dados do *United States Patent and Trademark Office* (USPTO). (1) Sendo assim, a falta de tecnologia nacional cria um cenário onde os equipamentos de ressonância magnética utilizados no país são sistemas proprietários de grandes empresas estrangeiras e possuem alto custo, chegando a valores superiores a US\$ 1.000.000,00. Embora na grande maioria dos casos a parte mais significativa do custo do equipamento seja devido ao magneto utilizado, outros gastos na etapa de pós-venda também são elevados como, por exemplo, a manutenção do *hardware* e as atualizações de *software*. Como consequência, apenas clínicas ou centros de pesquisa de grande porte possuem condições para adquirir um equipamento de RM, justificando parte da baixa produção mencionada anteriormente.

Outro fator relevante é a falta de flexibilidade e a dificuldade de se intervir nos sistemas comerciais que, por sua vez, afeta principalmente grupos com linhas de pesquisas relacionadas ao desenvolvimento de novos métodos de RM. Um método de sequência de pulsos de radio-frequência (RF) de ressonância magnética, ou simplesmente método de RM, é caracterizado pela definição de operações em nível de *hardware* e *software* que permitem a realização de um determinado experimento. Ele define a sequência de pulsos de RF que será aplicada sobre amostra ou paciente, a maneira com que os dados recebidos serão processados e organizados, além de prover um conjunto de parâmetros, e seus respectivos critérios de validação, que controlam diversas características do experimento. Deste modo, o desenvolvimento de novos métodos exige um conhecimento e controle muito grande do equipamento como um todo. Além disso, a importância das pesquisas nesta área está diretamente ligada à descoberta de novas aplicações ou aperfeiçoamento das mesmas como,

por exemplo, reduzir o tempo de duração do experimento ou exame clínico, melhoria da qualidade de uma imagem, redução de artefatos, novos meios de diagnósticos, caracterização de novos materiais, *etc.*

A modificação de métodos já existentes, por menor que seja, pode demandar muito tempo, inviabilizando o desenvolvimento ágil. Além disso, a dificuldade técnica para que se possam realizar estas tarefas fazem com que novos gastos com cursos, que normalmente são ministrados pelos próprios fabricantes, sejam necessários. Alguns fatores contribuem com a existência destes problemas como, por exemplo, a dependência da estratégia de venda do fabricante, a influência de órgãos que regulamentam a fabricação de produtos para uso clínico, que ocorre principalmente com equipamentos de Imagem por Ressonância Magnética (IRM) e o uso de uma arquitetura proprietária e fechada, onde se ocultam os detalhes de implementação.

Do ponto de vista dos avanços tecnológicos, há algum tempo já se falava a respeito da analogia entre o que ocorreu com os computadores com o que poderia acontecer com os equipamentos de RM. (2) Em sua primeira geração, os computadores construídos utilizando válvulas eletrônicas a vácuo, pesavam dezenas de toneladas, ocupavam alguns milhares de metros quadrados e conseguiam realizar apenas 40.000 operações por segundo aproximadamente. (3) Nos dias de hoje, os computadores de uso pessoal, incluindo *desktops*, *notebooks*, *tablets* e até mesmo *smartphones* possuem poder computacional superior, menor custo, menor tamanho e o seu uso foi estendido a todos os tipos de usuários.

Nos últimos tempos a instrumentação para ressonância magnética tem passado por um cenário semelhante. Ela foi e continua sendo impulsionada pelos avanços tecnológicos e também pela criação de novos métodos de RM (particularmente fMRI). (4) Em especial, novas tecnologias oriundas do conceito de hardware reconfigurável tem causado um forte impacto na área como é o caso da *Field-Programmable Gate Array* (FPGA). Uma FPGA é um arranjo de portas lógicas que podem ser programadas através de uma linguagem de descrição de hardware conhecida como *Hardware Description Language* (HDL), ao contrário da eletrônica convencional, onde todas as funcionalidades do *chip* são definidas durante a fabricação. Através desta tecnologia é possível produzir circuitos lógicos simples como portas *AND* e *OR*, circuitos combinacionais complexos, microprocessadores, *etc.* Alguns trabalhos recentes mostram a criação de equipamentos de RM utilizando FPGAs. (5-7)

No que tange à evolução do *software* nas diversas camadas, ou seja, nos avanços dos sistemas operacionais, das linguagens de programação e respectivos compiladores, sistemas gerenciadores de banco de dados, bibliotecas, aplicações, entre outros, o cenário não é

diferente. A consolidação da engenharia de *software*, a disseminação de informação através da *World Wide Web* (WWW), o aumento das comunidades de *software* livre e o progresso da ciência da computação em si são alguns dos fatores que influenciaram estes progressos. Sob a ótica das aplicações voltadas à ciência, incluindo RM, uma variedade muito grande de ferramentas para diversas finalidades existem, onde muitas delas são gratuitas. No entanto, o uso de ferramentas específicas tem como consequência a restrição do conjunto de funcionalidades, que é definida pelos desenvolvedores da mesma, onde, em alguns casos, até mesmo pequenas alterações são inviáveis. Algumas aplicações *open source*, ou seja, que possuem seu código fonte disponível, permitem que se possa realizar qualquer tipo de modificação. Contudo, estas modificações estão sujeitas à qualidade da documentação da ferramenta e, na maioria dos casos, um conhecimento aprofundado das tecnologias utilizadas. Sendo assim, é necessário recorrer ao desenvolvimento de ferramentas próprias.

Se por um lado o desenvolvimento de *software* pode trazer vantagens em relação à criação ou personalização de novas funcionalidades, alguns fatores devem ser considerados antes de dar início a um projeto de *software*. Entre eles, a escolha das tecnologias utilizadas pode ser crucial, tendo em vista que elas estão diretamente ligadas ao retardamento da obsolescência, à disponibilidade de recursos humanos que possuam os devidos conhecimentos, à portabilidade, ao tempo e ao custo necessários para o desenvolvimento do projeto. Outros fatores como especificação, documentação, validação e testes são importantes e também podem aumentar significativamente o custo do projeto.

Particularmente, um conjunto de tecnologias em nível de *software* tem chamado atenção da comunidade científica. A linguagem de programação *Python*, em conjunto com a biblioteca para cálculos numéricos, *Numpy*, e a biblioteca de visualização gráfica, *Matplotlib*, resultaram em um ambiente de programação poderoso, que substitui ferramentas bem consolidadas, como é o caso do *software* proprietário *Matlab*. (8-11) O *Python*, embora seja uma linguagem interpretada e dinâmica, permite que trechos de códigos escrito em C e C++ sejam integrados por meio de *wrapping*, ou empacotamentos, possibilitando o aumento de performance em regiões críticas de desempenho ou utilização de algoritmos existentes nestas linguagens. Todavia, o uso da biblioteca *Numpy* já possui a maioria de suas funcionalidades utilizando este recurso, permitindo o desenvolvimento de operações numéricas diretamente em linguagem *Python*, que possui sintaxe simplificada e alta legibilidade com algumas otimizações de desempenho. Além disso, um tipo de dado existente na *Numpy*, o *Numpy Array*, que nada mais é do que uma coleção uniforme de elementos, onde cada um deles ocupa o mesmo espaço em memória, permite a operação vetorial sobre os dados de maneira a

eliminar o problema de desempenho. Outra ferramenta que possui uma série de funcionalidades voltadas ao processamento de sinais e também é baseada nos recursos da *Numpy* é a biblioteca *Scipy*.

Devido ao caráter de propósito geral da linguagem *Python*, é possível criar aplicações voltadas ao usuário final utilizando-se de recursos de interface gráfica, também conhecida como *Graphical User Interface* (GUI). Em um conjunto razoável de opções, algumas bibliotecas que permitem o uso deste recurso são: *PyGtk*, *TkInter*, *PyQt* e *PySide*. (12-15) Em particular, as duas últimas bibliotecas mencionadas realizam o papel de encapsular outra biblioteca existente, o *Qt*. O *Qt* constitui a base do *KDE* e é um arcabouço de plataforma cruzada, ou seja, compatível em diversos sistemas operacionais, que permite o desenvolvimento rápido de interfaces gráficas para usuário e é originalmente implementado e disponível para utilização em linguagem de programação C++. (16-17) Sendo assim, as bibliotecas *PySide* e *PyQt* permitem então a utilização do *Qt* em uma aplicação desenvolvida em *Python*.

Este trabalho propõe o desenvolvimento de ferramentas de *software* que possam permitir a operação do espectrômetro completamente digital de RM do CIERMag. Este espectrômetro utiliza a abordagem de *hardware* reconfigurável e está projetado para atuar nas mais diversas aplicações, incluindo a tomografia. De um modo geral, as ferramentas desenvolvidas durante a realização deste trabalho devem prover uma interface entre os usuários e o equipamento, permitindo a utilização de métodos de sequências de pulsos previamente desenvolvidos.

Os sistemas de aquisição, pré-processamento, organização e visualização dos dados constituem o *ToRM Console*. Além disso, uma *Application Programming Interface* (API), o *ToRM Processing API*, para a realização de pós-processamento foi desenvolvida e será descrita neste trabalho. Esta última, pode ser integrada ao *ToRM Console* ou ser executada independentemente. Do ponto de vista conceitual, este trabalho é baseado na necessidade de se obter partes de *software* fundamentais para o funcionamento de um equipamento de ressonância magnética flexível, robusto e acessível apoiando-se em tecnologias inovadoras mencionadas anteriormente.

No próximo capítulo são discutidos os princípios físicos básicos de RM necessários para o entendimento completo desta técnica. Em seguida, no capítulo 3, são abordados todos os elementos que constituem o espectrômetro desenvolvido pelo CIERMag, com a finalidade de apontar a conexão entre as ferramentas desenvolvidas pelo grupo com as aplicações que foram desenvolvidas no decorrer deste trabalho. Os capítulos 4 e 5 demonstram as etapas de

desenvolvimento e as funcionalidades dos subsistemas *ToRM Console* e *ToRM Processing API*, respectivamente. No capítulo 6, a operação do sistema é mostrada através da utilização de dados sintetizados. Além disso, a realização de um experimento para estudo do comportamento do tempo de relaxação transversal de soluções de sulfato de cobre em função da variação da concentração é mostrada através das ferramentas desenvolvidas. Por fim, as conclusões e limitações deste trabalho estão contidas no capítulo 7, com a inclusão de sugestões para trabalhos futuros.

2 PRINCÍPIOS DE RESSONÂNCIA MAGNÉTICA

2.1 Introdução

Durante o ano de 1922, a realização do experimento de Stern-Gerlach demonstrou a existência de uma propriedade quântica dos elétrons e átomos, também denominada por *spin*, que é uma forma intrínseca de momento angular e não possui análogo clássico. (18) Em 1930, utilizando uma versão aprimorada do equipamento de Stern-Gerlach, Isidor Isaac Rabi mostrou ser possível a mudança dos estados dos momentos magnéticos através da aplicação de um campo magnético variável no tempo, obtendo o prêmio Nobel no ano de 1944. Em 1946, através do trabalho que também rendeu o prêmio Nobel aos autores Bloch e Purcell no ano de 1952, foi possível observar o efeito da precessão dos spins em torno de um campo magnético, bem como determinar a base experimental e teórica da técnica de ressonância magnética nuclear.

Neste capítulo os princípios físicos que regem a ressonância magnética nuclear serão abordados. Na subseção 2.2.1 serão introduzidas algumas grandezas físicas que são fundamentais para a compreensão desta técnica. Alguns tópicos de mecânica quântica foram utilizados apenas para justificar alguns fenômenos que não podem ser explicados totalmente pela mecânica clássica. Sendo assim, uma abordagem semi-clássica será utilizada de maneira a simplificar a descrição dos princípios físicos de modo suficientemente abrangente para o entendimento deste trabalho. Embora este trabalho lide com alguns aspectos de formação de imagens através da técnica de ressonância de magnética, não serão abordados os princípios que regem esse tipo de método, que é discutido com detalhes em (19).

2.2 Princípios físicos

2.2.1 Momento angular, *spin* e momento magnético.

O momento angular orbital, também chamado de *momentum* angular ou quantidade de movimento angular (ou de rotação) é uma grandeza clássica que quantifica a rotação de um corpo levando em consideração a sua distribuição de massa. Para o caso em que um objeto de massa m , possuindo dimensões espaciais muito menores do que a distância radial do seu eixo de rotação, ou seja, para o caso de um objeto pontual, o momento angular $\vec{L}$ é descrito através do produto vetorial entre a posição $\vec{r}$ em relação a uma determinada origem e o momento linear, ou quantidade de movimento, $\vec{p} = m\vec{v}$, de acordo com a equação abaixo:

$$\vec{L} = \vec{r} \times \vec{p} = \vec{r} \times m\vec{v} \quad (1)$$

Desta forma, a grandeza $\vec{L}$ é, portanto, uma grandeza vetorial e o seu sentido indica a direção em que a rotação ocorre quando a regra da mão direita é aplicada. De uma maneira mais geral, o momento angular pode ser calculado através da velocidade angular $\vec{\omega}$ e o momento de inércia I^i , ambos em torno de um determinado eixo de rotação:

$$\vec{L} = I\vec{\omega} \quad (2)$$

Se um objeto está em rotação devido à existência de forças externas, a grandeza vetorial $\vec{\tau}$, denominada torque, pode ser calculada através do produto vetorial da força externa resultante $\vec{F}$ com a distância $\vec{r}$ entre o ponto onde a força está sendo aplicada e o eixo de rotação. Além disso, é possível mostrar que a taxa variação temporal do momento angular é proporcional ao torque, de maneira que:

ⁱ O momento de inércia é, na verdade, uma grandeza tensorial, embora, neste caso, apenas uma de suas componentes esteja sendo utilizada explicitamente.

$$\vec{\tau} = \vec{r} \times \vec{F} = \frac{d\vec{L}}{dt} \quad (3)$$

Sendo assim, quando não há torque externo resultante, o momento angular é conservado, como uma extensão da segunda Lei de Newton $\vec{F} = d\vec{p}/dt$ para movimentos de rotação, onde $\vec{F}$ é substituído por $\vec{\tau}$ e $\vec{p}$ por $\vec{L}$:

$$\frac{d\vec{L}}{dt} = 0 = \vec{\tau} \leftrightarrow \vec{L} = \text{constante} \quad (4)$$

As equações 1-4 descrevem o comportamento clássico da grandeza momento angular. Deste modo, nenhuma restrição em relação ao módulo, sentido e origem de $\vec{L}$, além das impostas por estas equações são observadas. Do ponto de vista da mecânica quântica, o observável do *momentum* angular orbital $\mathbf{L}$, que é formado pelas componentes $\mathbf{L}_x$, $\mathbf{L}_y$ e $\mathbf{L}_z$ é um equivalente da grandeza clássica $\vec{L}$ e possui as seguintes restriçõesⁱⁱ:

$$\mathbf{L}^2 |l, m\rangle = l(l+1)\hbar |l, m\rangle \quad \mathbf{L}^2 = \mathbf{L}_x^2 + \mathbf{L}_y^2 + \mathbf{L}_z^2 \quad (5)$$

$$\mathbf{L}_z |l, m\rangle = m\hbar |l, m\rangle \quad m = -l, -l+1, \dots, l-1, l \quad (6)$$

Onde $\hbar$ é a constante de Planck dividida por 2π e l e m são os números quânticos associados à $\mathbf{L}^2$ e $\mathbf{L}_z$, respectivamente.

Deste modo, apenas o conjunto discreto de autovalores $l(l+1)\hbar$ e $m\hbar$ para $\mathbf{L}^2$ e $\mathbf{L}_z$, respectivamente, são permitidos, fazendo com que existam restrições no módulo e direção do *momentum* angular. Além disso, são permitidos valores inteiros e semi-inteiros para l . Se, por exemplo, a partícula possui *momentum* angular $l = 1/2$, então os valores possíveis de m serão $-1/2$ e $+1/2$, de modo que apenas dois valores para a componente $\mathbf{L}_z$ sejam possíveis. A figura 1 ilustra este resultado para o caso $l = 1$.

ⁱⁱ Em função das regras de comutação entre os operadores $\mathbf{L}_x$, $\mathbf{L}_y$ e $\mathbf{L}_z$ é possível medir apenas uma componente cartesiana ($\mathbf{L}_z$, apenas por convenção) e o módulo $\mathbf{L}^2$ simultaneamente. Sendo assim, é vantajoso descrever o sistema em função da base $|l, m\rangle$, que são autovetores simultâneos dos observáveis $\mathbf{L}^2$ e $\mathbf{L}_z$.

Figura 1 - Valores possíveis para a componente L_z do *momentum* angular para o caso $l = 1$.

Fonte: Elaborada pelo autor.

Outro fato importante oriundo da mecânica quântica é a existência de uma das formas de *momentum* angular que é intrínseca a uma partícula ou átomo. A sua observação surgiu durante a realização do experimento de Stern e Gerlach e a sua quantização possui as mesmas características do *momentum* angular orbital. Desta forma, é interessante definir o observável do *momentum* angular total, denominado por J , que é caracterizado pela soma do *momentum* angular orbital L e do *momentum* angular intrínseco S (*spin*):

$$J = L + S \quad (7)$$

Até o presente momento, as grandezas físicas foram abordadas sem levar em consideração os tipos de partículas que realmente são de interesse do ponto de vista da ressonância magnética nuclear. O termo ‘nuclear’ⁱⁱⁱ, que está contido no nome desta técnica, está diretamente ligado à observação deste fenômeno físico sob o ponto de vista dos núcleos atômicos. Os núcleos são formados por nêutrons e prótons e ambos possuem *spin* $1/2$.

Outra propriedade importante é o momento de dipolo magnético, também conhecido como momento magnético. A relação entre o momento magnético de uma partícula com seu *spin* é dado pela equação abaixo:

$$\mu = \gamma S \quad (8)$$

Onde a constante γ é a razão giromagnética e possui valores diferentes para cada tipo de partícula. A diferença entre a razão giromagnética entre os constituintes do núcleo e o elétron é o principal motivo para observação apenas dos núcleos no contexto da ressonância

ⁱⁱⁱ A sigla RM, omitindo-se o termo ‘nuclear’ é normalmente utilizada atualmente para evitar confusões com o fenômeno de decaimento nuclear de outras técnicas que utilizam radiação ionizante.

magnética nuclear^{iv}. Entretanto, outra técnica denominada Ressonância Paramagnética Eletrônica (RPE) estuda este fenômeno do ponto de vista do comportamento dos elétrons, mas não fará parte do escopo deste trabalho. (20)

No caso dos núcleos atômicos, apenas uma partícula, isto é, um próton ou um nêutron, é responsável pelo *spin* resultante e, conseqüentemente, pelo momento magnético do núcleo. Seja Z o número atômico de um determinado elemento e M o número de massa, então o comportamento do *spin* e o momento magnético resultante será de acordo com a tabela abaixo: (21)

Tabela 1 - Determinação do momento magnético nuclear em função da paridade do número de massa (M) e número atômico (Z).

Tipo	Z	M	μ
1	Par	Par	Nulo
2	Ímpar	Ímpar	Determinado pelo próton excedente
3	Par	Ímpar	Determinado pelo nêutron excedente
4	Ímpar	Par	Não é possível realizar uma única interpretação ^v

Fonte: Alfred Landé, 1934 (21).

Para ilustrar uma das consequências da existência do momento angular intrínseco, considere um sistema de *spin* $\frac{1}{2}$ e momento magnético $\vec{\mu}$. Se esta partícula estiver sujeita a um campo magnético $\vec{B}_0 = B_0 \hat{z}$, então a energia potencial clássica será:

$$W = -\vec{\mu} \cdot \vec{B}_0 = -\mu_z B_0 = -\gamma B_0 L_z = \omega_0 L_z \quad (9)$$

A frequência ω_0 é conhecida como frequência de Larmor e será discutida adiante.

$$\omega_0 = -\gamma B_0 \quad (10)$$

^{iv} A razão giromagnética do elétron é 658 vezes maior do que a do próton.

^v Estes casos são raros e ocorrem apenas nas moléculas H_2^1 , Li_6^3 , B_{10}^5 e N_{14}^7 .

Levando em consideração apenas a quantização dos graus internos de liberdade, o momento angular orbital pode ser desconsiderado de maneira que $\mathbf{J} = \mathbf{S}$. Sendo assim, o Hamiltoniano deste sistema pode ser escrito como:

$$\mathbf{H} = \omega_0 \mathbf{S}_z \quad (11)$$

Os autovalores desta equação, que serão os dois níveis de energia possíveis, podem ser obtidos através da solução da equação de Schrödinger, uma vez que ela não possui dependência temporal. Os autovetores de $\mathbf{H}$ serão os mesmos de S_z , $|\uparrow\rangle$ e $|\downarrow\rangle$, de maneira que:

$$\mathbf{H} |\uparrow\rangle = +\frac{\hbar\omega_0}{2} |\uparrow\rangle \quad (12)$$

$$\mathbf{H} |\downarrow\rangle = -\frac{\hbar\omega_0}{2} |\downarrow\rangle \quad (13)$$

Desta forma, existem dois níveis energias $E_+ = +\hbar\omega_0/2$ e $E_- = -\hbar\omega_0/2$, separados por uma quantidade de energia igual a $\hbar\omega_0$.

2.2.2 Magnetização e comportamento de um próton sujeito a um campo magnético.

Um próton de momento magnético μ sob a aplicação de um campo magnético $\vec{B}$ está sujeito a um torque, de maneira que:

$$\vec{\tau} = \vec{\mu} \times \vec{B} \quad (14)$$

A taxa de variação temporal do *momentum* angular pode ser, então, calculada através da equação 3:

$$\frac{d\vec{L}}{dt} = \vec{\mu} \times \vec{B} \quad (15)$$

Utilizando uma analogia clássica para a equação 8, o momento magnético pode ser relacionado com o *momentum* angular $\vec{L}$, de modo que:

$$\frac{d\vec{\mu}}{dt} = \gamma \vec{\mu} \times \vec{B} \quad (16)$$

A figura 2 descreve o comportamento do momento magnético de acordo com a evolução temporal imposta pela equação 16.

Figura 2 - comportamento do momento magnético $\vec{\mu}$ sujeito a um campo magnético $\vec{B}$, conforme a equação (16).

Fonte: Adaptada de E. M. Haacke et al, 1999 (21).

De acordo com a figura 2, o módulo da variação espacial infinitesimal do momento magnético $d\vec{\mu}$ está relacionado com a variação angular $d\phi$ da seguinte forma:

$$|d\vec{\mu}| = \mu \sin\theta |d\phi| \quad (17)$$

Sendo assim, através da equação 16:

$$|d\vec{\mu}| = \mu \sin\theta |d\phi| = \gamma |\vec{\mu} \times \vec{B}| dt = \gamma \mu B \sin\theta dt \quad (18)$$

A velocidade angular pode, então, ser obtida através da razão giromagnética, γ , e da intensidade do campo magnético, B . O sentido da rotação pode ser obtido através da regra da mão direita.

$$\frac{|d\vec{\mu}|}{dt} = \gamma B = \omega \quad (19)$$

Do ponto de vista experimental, outra grandeza será necessária para descrever a evolução de um sistema de prótons do ponto de vista macroscópico. Se considerarmos um volume V , denominado *voxel*, de maneira que em qualquer região do seu interior o campo

magnético possa ser considerado aproximadamente constante, a densidade de momento magnético $\vec{M}$, denominada magnetização, poderá ser definida da seguinte forma:

$$\vec{M} = \frac{1}{V} \sum_V \vec{\mu}_i \quad (20)$$

Este somatório deve ser efetuado sobre todos os prótons que compõem o sistema no interior do *voxel*. Embora, como visto anteriormente, o momento magnético possua valores quantizados em função de sua dependência com o *spin*, a magnetização pode ser considerada uma grandeza contínua. Esta afirmação é baseada no fato de que os sistemas de interesse possuem uma quantidade de prótons da ordem do número de Avogrado.

Desta forma, através das equações 16 e 20 podemos escrever a evolução temporal da magnetização da seguinte forma:

$$\frac{d\vec{M}}{dt} = \gamma \vec{M} \times \vec{B} \quad (21)$$

Se o sistema estiver com temperatura igual ao zero absoluto haverá alinhamento total dos *spins* com o campo magnético de modo que a magnetização atingirá o maior valor possível. Por outro lado, se nenhum campo for aplicado sobre o *voxel*, todos os *spins* estarão aleatoriamente alinhados e, portanto, a densidade de momento magnético será nula. No caso intermediário, que é o de interesse para os estudos de RM, o *voxel* está contido em um reservatório térmico com temperatura T sujeito a um campo magnético externo de intensidade B_0 . Sendo assim, devido aos efeitos de agitação térmica, as orientações paralelas e antiparalelas ao campo magnético são possíveis, embora haja uma fração excedente de *spins*, da ordem de $\mu B_0 / k_B T$, alinhados ao campo magnético, ou seja, possuindo estados com a menor energia. Se o número de prótons do sistema for N , então essa magnetização de equilíbrio, M_0 , será, portanto, da ordem de:

$$M_0 = N \mu \frac{\mu B_0}{k_B T} = N \frac{\mu^2 B_0}{k_B T} \quad (22)$$

Onde k_B é a constante de Boltzmann.

2.2.3 Mecanismos de relaxação e equação de Bloch

A equação 21 descreve corretamente a evolução do vetor da magnetização quando não há interações entre os *spins* e o meio e nem mesmo interações *spin-spin*. No entanto, o fenômeno conhecido por relaxação pode ser observado quando a magnetização é retirada da sua condição de equilíbrio, indicando a existência dessas interações. A magnetização longitudinal, ou seja, a componente de densidade de momento magnético na direção paralela ao campo magnético tende a voltar ao seu valor de equilíbrio, M_0 , visto na equação 22, exponencialmente com tempo característico denominado *tempo de relaxação longitudinal*, T_1 . De maneira análoga, a magnetização transversal à direção do campo magnético tende a se tornar nula exponencialmente com o tempo característico denominado *tempo de relaxação transversal*, T_2 .

Os fenômenos de relaxação longitudinal e transversal ocorrem independentemente e a diferença entre os tempos de decaimento é dada pela origem física de cada um deles. No caso da relaxação longitudinal predomina-se o efeito da interação *spin-rede*, enquanto que a relaxação transversal é causada pela interação *spin-spin*. Além disso, $T_1 > T_2$ no caso geral.

Sendo assim, deve-se realizar a adaptação da equação 21 para que o fenômeno de relaxação seja levado em consideração:

$$\frac{d\vec{M}}{dt} = \gamma \vec{M} \times \vec{B} - \frac{1}{T_1} (M_0 - M_z) \hat{z} - \frac{\vec{M}_\perp}{T_2} \quad (23)$$

Onde $\vec{M}_\perp$ é a componente transversal ao campo magnético aplicado. Esta equação é a base do fenômeno de RM utilizando um arcabouço semi-clássico e é denominada *equação de Bloch*. Se o campo magnético for $\vec{B} = B_0 \hat{z}$, através da realização do produto vetorial contido em 23 é possível obter as componentes da equação de Bloch, de maneira que:

$$\frac{dM_z}{dt} = \frac{M_0 - M_z}{T_1} \quad (24)$$

$$\frac{dM_x}{dt} = \omega_0 M_y - \frac{M_x}{T_2} \quad (25)$$

$$\frac{dM_y}{dt} = -\omega_0 M_x - \frac{M_y}{T_2} \quad (26)$$

Onde $\omega_0 = -\gamma B_0$ é a frequência de Larmor, com que os momentos magnéticos precessionam, definida na equação 10. A solução para cada uma dessas componentes cartesianas quando os únicos componentes de campo magnético variável são os correspondentes ao campo local produzido pelos próprios momentos magnéticos na região de seus vizinhos será:

$$M_x(t) = e^{-t/T_2} (M_x(0)\cos \omega_0 t + M_y(0)\sin \omega_0 t) \quad (27)$$

$$M_y(t) = e^{-t/T_2} (M_y(0)\cos \omega_0 t - M_x(0)\sin \omega_0 t) \quad (28)$$

$$M_z(t) = M_z(0)e^{-t/T_1} + \left(1 - e^{-t/T_1}\right) M_0 \quad (29)$$

$M_x(0)$, $M_y(0)$ e $M_z(0)$ são os valores iniciais ($t = 0$) das componentes cartesianas da magnetização e M_0 é a magnitude do valor de equilíbrio da magnetização.

Quando a magnetização é retirada do seu ponto de equilíbrio, após um tempo longo, a magnetização tende a voltar a sua condição de equilíbrio, de modo que:

$$M_z(\infty) = M_0; M_x(\infty) = 0; M_y(\infty) = 0 \quad (30)$$

A figura 3 exemplifica o comportamento da magnetização, de acordo com as equações 27, 28 e 29.

Figura 3 - Evolução temporal da magnetização, levando em consideração os mecanismos de relaxação. Inicialmente, toda magnetização encontra-se no plano transversal. Esta situação pode ser obtida com a aplicação de um pulso de radiofrequência, que será vista na subseção 2.2.5. Em seguida, ela retorna a sua condição de equilíbrio, conforme previsto pela equação de Bloch.

Fonte: Elaborada pelo autor.

2.2.4 Detecção de sinal

Foi visto até o momento que, quando um campo magnético uniforme é aplicado sobre um sistema de *spins* composto por prótons, a magnetização resultante tende a um valor constante, M_0 , na mesma direção em que o campo, convencionalmente definida por direção z . Sendo assim, o primeiro componente de um sistema de RM é uma fonte de um campo magnético que, dependendo da aplicação pode ser um magneto supercondutor, permanente ou resistivo.

A primeira etapa de um experimento de RM seria, então, a aplicação de um pulso de radiofrequência com o intuito de remover a magnetização resultante do seu equilíbrio, ‘excitando’ a amostra. Para que isso seja possível, uma bobina de radiofrequência,

denominada bobina transmissora, é normalmente utilizada. Para que seja possível a excitação nas direções x e y (direção transversal) podem-se utilizar duas bobinas perpendiculares entre si e transversais ao campo magnético. Estas mesmas bobinas podem ser utilizadas para receber os sinais oriundos da amostra após a excitação e, quando isto ocorre, estas bobinas são denominadas transeptoras, por realizarem o papel de receptoras e transmissoras simultaneamente, dando origem ao nome. Para a compreensão do funcionamento do sistema de aquisição, serão abordados a seguir os princípios físicos necessários.

É possível mostrar, através do *princípio da reciprocidade*, que o fluxo Φ_M através da bobina receptora causada por um sistema com magnetização total $\vec{M}(\vec{r}, t)$ pode ser calculado em função do campo magnético $\vec{B}^{rec}(\vec{r})$, que seria produzido pela bobina em uma posição $\vec{r}$ no interior de uma amostra de volume V, de maneira que a que força eletromotriz gerada na região onde está localizada a bobina possa ser calculada da seguinte forma:

$$f.e.m. = -\frac{d\Phi_M}{dt} = -\frac{d}{dt} \int_V d^3r \vec{B}^{rec}(\vec{r}) \cdot \vec{M}(\vec{r}, t) \quad (31)$$

Deste modo, podemos estimar o sinal captado na bobina em função das componentes cartesianas de $\vec{M}(\vec{r}, t)$ e $\vec{B}^{rec}(\vec{r})$:

$$S \propto -\frac{d}{dt} \int_V d^3r \left(B_x^{rec} M_x(\vec{r}, t) + B_y^{rec} M_y(\vec{r}, t) + B_z^{rec} M_z(\vec{r}, t) \right) \quad (32)$$

As componentes $M_x(t)$ e $M_y(t)$ da solução da equação Bloch, são utilizadas para constituir a componente transversal, e podem ser escritas em notação complexa, de maneira a facilitar os cálculos das derivadas temporais, além de proporcionar maiores informações a respeito da fase.

$$M_+(t) = M_x(t) + iM_y(t) \quad (33)$$

$$M_+(t) = e^{-t/T_2} e^{-i\omega_0 t} M_+(0) \quad (34)$$

A equação 32 pode ser simplificada através das seguintes considerações: as derivadas dos termos e^{-t/T_1} e e^{-t/T_2} podem ser desprezadas em função da comparação entre os valores

de $1/T_1$ e $1/T_2$ com o valor de ω_0 e, portanto, a componente longitudinal $M_z(\vec{r}, t)$ poderá ser totalmente desprezada, ainda que haja campo na direção z do campo $\vec{B}^{rec}(\vec{r})$. Estas aproximações não são válidas para sólidos, onde T_2 pode ser da ordem de microssegundos.

A expressão final, levando em consideração as aproximações mencionadas, será então:

$$S \propto \omega_0 \int_V d^3r e^{-t/T_2} [B_x^{rec}(\vec{r}) \text{Re}\{iM_+(\vec{r}, 0)e^{-i\omega_0 t}\} + B_y^{rec}(\vec{r}) \text{Im}\{iM_+(\vec{r}, 0)e^{-i\omega_0 t}\}] \quad (35)$$

$$S \propto \omega_0 \int_V d^3r e^{-t/T_2} M_\perp(\vec{r}, 0) [B_x^{rec}(\vec{r}) \sin(\omega_0 t - \phi_0(\vec{r})) + B_y^{rec}(\vec{r}) \cos(\omega_0 t - \phi_0(\vec{r}))] \quad (36)$$

A dependência direta de $M_\perp(\vec{r}, 0)$ na equação 8 mostra o porquê do módulo da magnetização transversal ser considerado como sendo o ‘sinal’ de RM. Além disso, a contribuição do valor de ω_0 à amplitude do sinal garante a possibilidade de ele ser observado.

Sendo transceptor ou apenas receptor, o sistema que engloba as bobinas de leitura possui um papel adicional durante a etapa de aquisição. Como a frequência do sinal obtido é na faixa da radiofrequência, a amostragem pode trazer alguns problemas. Entre eles, a quantidade excessiva de pontos a serem digitalizados sem perda de informação, ou seja, obedecendo ao teorema da amostragem, também conhecido por critério de Nyquist. Entretanto, como normalmente a banda da região espectral de interesse normalmente é inferior à ordem de grandeza da radiofrequência do sinal obtido, ou seja, apenas uma pequena porção do espectro de frequências contém informações relevantes em um experimento, é desejável, então, obter apenas esta parte.

A técnica utilizada para que isso seja possível é denominada ‘demodulação’ e é semelhante ao observar a evolução da magnetização resultante a partir de um referencial de coordenadas girantes com frequência Ω em torno do eixo z . Além de permitir a redução da quantidade de dados adquiridos, através dessa técnica é possível determinar simultaneamente as componentes transversais da magnetização M_x e M_y e, portanto, obter as informações da magnitude e fase da magnetização transversal total M_+ .

A frequência base Ω , como é denominada, diverge ligeiramente da frequência de Larmor, de maneira que $\Omega = \omega_0 + \delta\omega$. Tipicamente a frequência de Larmor é da ordem de dezenas de MHz e o valor de $\delta\omega$ é da ordem de kHz.

Considerando um sinal senoidal obtido de frequência ω_0 , ou seja, com frequência de Larmor e o multiplicando por $\text{sen}(\Omega t)$, então o resultado obtido será, por definição, considerado como sendo o sinal de entrada do *canal real*:

$$\text{sen}(\omega_0 t)\text{sen}(\Omega t) = \frac{\cos(\delta\omega t) - \cos([\Omega + 2\omega_0 + \delta\omega]t)}{2} \quad (37)$$

Analogamente, teremos a entrada do *canal imaginário* se este mesmo sinal for multiplicado por $\cos(\Omega t)$:

$$\text{sen}(\omega_0 t)\cos(\Omega t) = \frac{\text{sen}(\delta\omega t) + \sin([\Omega + 2\omega_0 + \delta\omega]t)}{2} \quad (38)$$

Após passar por uma filtragem, os canais reais e imaginários terão apenas as componentes de mais baixa frequência e, portanto, serão proporcionais aos seguintes valores.

$$S_{real} \propto \frac{\cos(\delta\omega t)}{2} \quad (39)$$

$$S_{imaginário} \propto \frac{\text{sen}(\delta\omega t)}{2} \quad (40)$$

E, portanto, as oscilações rápidas foram removidas através dessa transformação para um referencial girante. Para completar a abordagem do tratamento do sinal em fase e quadratura, utilizando notação complexa por conveniência, o sinal total demodulado pode ser definido como:

$$s(t) \equiv S_{real} + iS_{imaginário} \quad (41)$$

Se os campos B_x^{rec} e B_y^{rec} forem escritos como função da magnitude transversal do campo $\vec{B}^{rec}$, $B_\perp$ e do ângulo Θ_B , teremos:

$$B_x^{rec} \equiv B_\perp \cos\theta_B \quad (42)$$

$$B_y^{rec} \equiv B_\perp \sin\theta_B \quad (43)$$

E, portanto, o sinal $s(t)$ poderá ser escrito da seguinte forma:

$$s(t) \propto \omega_0 \int_V d^3r e^{-t/T_2} M_{\perp}(\vec{r}, 0) e^{-i((\Omega - \omega_0)t + \phi_0 - \Theta_B)} \quad (44)$$

Se duas bobinas, em quadratura, ou seja, ortogonais entre si forem utilizadas para receber os sinais simultaneamente, então, poderá haver aumento da relação sinal-ruído por dois motivos: o sinal captado poderá ser até cerca 40% maior; e o fato do ruído gerado pelos dois receptores não serem correlacionados, embora este último seja apenas verdade quando o ruído não é gerado pelo acoplamento entre a amostra e o receptor. Além disso, os canais imaginários e reais são obtidos sem a necessidade de demodulação, levando em consideração que os sinais obtidos já estão geometricamente defasados de 90 graus e, portanto, possuem informação completa sobre a magnetização transversal, se for desconsiderado o fato das oscilações rápidas.

Depois que os sinais em fase e quadratura são obtidos, uma etapa adicional de filtragem pode ser necessária. Esta filtragem é feita, normalmente, quando se utiliza demodulação digital, como no caso do espectrômetro do CIERMag, que foi utilizado neste trabalho, através de um filtro FIR, ou seja, um filtro de resposta ao impulso finita e ele é utilizado para se obter apenas a região espectral de interesse, eliminando toda a informação presente na região removida, geralmente contendo ruído. Esta etapa pode ser realizada em *software*, *hardware* ou em ambos, levando em consideração de que este tipo de filtragem pode atuar em ‘cascata’, ou seja, com filtrações consecutivas.

2.2.5 Aplicação de pulsos e condição de ressonância

Na subseção 2.2.3, através da solução da equação de Bloch, foi demonstrado o determinismo contido no comportamento da magnetização de um sistema de prótons sujeito a um campo magnético constante B_0 . Além disso, na subseção anterior, foi discutido que os sinais eletromagnéticos produzidos pela magnetização resultante podem ser observados com o auxílio de uma bobina e que esta mesma bobina também pode ser utilizada para a aplicação dos pulsos de radiofrequência necessários para excitar o sistema. Nesta subseção serão apresentados brevemente os mecanismos de excitação.

O campo magnético $\vec{B}_1(t)$, ilustrado na figura 4, é responsável pela excitação. Além de ser ortogonal ao campo $\vec{B}_0$, este campo deve oscilar com frequência igual à frequência de

Larmor, $\omega_0 = -\gamma B_0$. Se o campo B_0 possuir intensidade de 1 Tesla, então, para o caso de núcleos de hidrogênio, a frequência de precessão dos momentos magnéticos e, portanto, a frequência com que o campo $\vec{B}_1(t)$ oscila será $f_0 = 2\pi\omega_0 = 42,6$ MHz.

Figura 4 - Esquema simplificado de uma bobina transmissora. O campo produzido pela bobina, $\vec{B}_1(t)$, aponta sempre ortogonalmente ao campo estático $\vec{B}_0$. Além disso, a frequência de oscilação deste campo deve ser igual à frequência de Larmor.

Fonte: Documentação interna do CIERMag.

No caso ilustrado pela figura 4, ainda que o campo magnético $\vec{B}_1(t)$ esteja sendo aplicado apenas na direção y, ele pode ser decomposto em duas componentes girando em sentidos opostos, ou seja, uma em sentido anti-horário e outra no sentido horário, com igual intensidade. Se a intensidade de cada uma dessas componentes for igual a B_1 , então o campo resultante será $\vec{B}_1(t) = 2B_1\cos(\omega_0 t)$. A figura abaixo ilustra esta decomposição.

Figura 5 - O campo oscilatório de intensidade $\vec{B}_1(t) = 2B_1\cos(\omega_0 t)$ pode ser decomposto em duas componentes, B_1^+ e B_1^- . Quando observadas em um referencial de coordenadas girantes, que gira em torno do eixo z com frequência de Larmor, apenas uma dessas componentes é efetivamente observada, levando em consideração somente os efeitos de primeira ordem.

Fonte: Fernando F. Paiva, 2004 (45).

Se este efeito for observado de um referencial que gira em torno do eixo z com frequência de Larmor, denominado *referencial de coordenadas girantes*, uma das componentes será estática, enquanto a outra será observada com frequência angular duas

vezes maior. A componente mais rápida pode ser desprezada, em função de não haver nenhuma contribuição em primeira ordem para o campo observado no referencial de coordenadas girantes. Como o campo magnético B_0 não é observado, o campo magnético resultante neste referencial será estático e de intensidade B_1 . Deste modo, após a aplicação deste campo durante um intervalo τ , a magnetização resultante sofrerá a rotação de um ângulo θ , tal que:

$$\theta = \gamma B_1 \tau \quad (45)$$

Fazendo com que as novas componentes, longitudinal e transversal, da magnetização sejam:

$$M_z = M_0 \cos \theta \quad (46)$$

$$M_{\perp} = M_0 \sin \theta \quad (47)$$

A figura 6 ilustra o comportamento da magnetização após a aplicação de distintos pulsos.

Figura 6 - Após a aplicação de um pulso com duração τ , a magnetização é deslocada de um ângulo $\theta = \gamma B \tau$ em torno do campo magnético B que, neste caso, aponta para a direção x (esquerda). Quando a magnetização é deslocada de um ângulo de 90 graus, utiliza-se o termo ‘aplicação de um pulso de $\pi/2$ ’ (centro). De maneira análoga, quando a magnetização sofre uma inversão no seu sentido, é utilizado um ‘pulso de π ’ (direita).

Fonte: Documentação interna do CIERMag.

2.2.6 Técnicas de aquisição: Decaimento livre da indução, T_2^* e *spin-eco*

O experimento mais simples de ressonância magnética consiste na aplicação de um pulso de $\pi/2$ e, imediatamente após, na obtenção do sinal. Desta forma, será observado o decaimento livre da indução sistema, mais conhecido como FID (*Free Induction Decay*), conforme a figura 7.

Figura 7 - Decaimento livre da indução (FID) em azul. A envoltória vermelha representa a evolução do módulo da magnetização transversal. O tempo de relaxação T_2^* é utilizado em função da inhomogeneidade de campo presente nos experimentos reais.

Fonte: Elaborada pelo autor.

Algumas formas de ruído presente no sinal podem ser eliminadas com a utilização de médias, através da repetição de uma estrutura de pulsos fixa. No caso da aplicação de apenas um pulso de $\pi/2$, conforme discutido acima, um dos cuidados a ser tomado é que o tempo de repetição, denominado T_R , seja maior do que o tempo de relaxação longitudinal, T_1 , para que apenas os efeitos de relaxação transversal possam ser observados.

De fato, o valor observado experimentalmente para o tempo transversal difere do tempo T_2 neste caso. Isto ocorre em função da inhomogeneidade, causada por imperfeições na fonte de campo de magnético, pela amostra ou pelo acoplamento entre eles, fazendo com que os núcleos recebam campos ligeiramente distintos, de modo que o mecanismo de relaxação transversal sofra variações. O tempo efetivo medido é denominado T_2^* e se relaciona com o tempo de relaxação transversal verdadeiro, T_2 , da seguinte forma:

$$\frac{1}{T_2^*} = \frac{1}{T_2'} + \frac{1}{T_2} \quad (48)$$

Onde a contribuição causada pela inhomogeneidade de campo está contida em T_2' e o tempo de relaxação T_2 contém apenas os efeitos termodinâmicos. Sendo assim, o tempo de decaimento observado no FID, caso o campo não seja perfeitamente homogêneo será dado pela equação acima. Desta forma, caso o tempo de relaxação transversal T_2 necessite ser medido, outra estratégia para aquisição deve ser utilizada. O método de aquisição *spin-eco* pode ser utilizado neste contexto.

Como os núcleos estão sujeitos a campos ligeiramente distintos, as frequências de Larmor serão aleatoriamente diferentes, de modo que haja perda de coerência de fase nos momentos magnéticos e, conseqüentemente, uma diminuição do valor da magnetização, reduzindo o tempo de relaxação transversal, conforme discutido. No entanto, se, após a aplicação do pulso de excitação de $\pi/2$ na direção x , for aplicado um pulso de π na direção y^{vi} após um tempo T_E , denominado tempo ao eco, a fase da magnetização será invertida de maneira que a coerência de fase seja novamente estabelecida no tempo $2T_E$, onde é realizada a leitura.

A amplitude da magnetização inicial, M_0 , é reduzida exponencialmente no tempo $2T_E$ com tempo de relaxação T_2 , isto é $M_{\perp}(t = 2T_E) = M_0 e^{-2T_E/T_2}$. A figura 8 abaixo descreve a técnica *spin-eco*.

^{vi} Um pulso de π também na direção x poderia ser utilizado.

Figura 8 - Evolução temporal da magnetização durante a execução de um método *spin-eco*. Inicialmente a magnetização está no seu estado de equilíbrio (A). Em seguida, um pulso de $\pi/2$ na direção $-x$ é aplicado, de maneira que a magnetização resultante aponte para direção y (B). O processo de relaxação transversal começa a atuar em suas duas formas simultaneamente: por meio de efeitos termodinâmicos; e por meio de inhomogeneidade de campo. A última faz com que os momentos magnéticos de cada um dos núcleos, representados fora de escala por 'a', 'b', 'c', 'd' e 'e', precessionem com frequências de Larmor diferentes, causando perda de coerência (C). Após um tempo T_E , a aplicação de um pulso de π na direção y faz com que as fases sejam invertidas, de modo que os momentos magnéticos comecem a precessionar com velocidade angular oposta a da situação anterior (D). No tempo $2T_E$, os momentos magnéticos entram novamente em coerência, maximizando o valor da magnetização. Nesta situação o decaimento do valor da magnetização deve-se somente aos efeitos termodinâmicos e, portanto, a leitura do sinal neste momento pode fornecer o valor do tempo de relaxação T_2 (E). Por fim, se não houver a aplicação de nenhum outro pulso, a magnetização transversal e longitudinal retornam aos seus respectivos valores de equilíbrio (F).

Fonte: Elaborada pelo autor.

3 ESPECTRÔMETRO DO CIERMAG

3.1 Introdução

Levando em consideração toda a experiência adquirida ao longo dos anos em que o grupo esteve envolvido com o desenvolvimento de tecnologias e equipamentos de ressonância magnética nuclear, algumas dificuldades observadas no passado foram essenciais para a motivação deste projeto como um todo. Entre elas, a forte dependência das partes que compunham a eletrônica do sistema fazia com que várias limitações no equipamento surgissem com o passar do tempo, inviabilizando a adequação do sistema aos novos métodos, técnicas durante a etapa de aquisição de sinais, entre outros fatores.

Este fato acompanha a grande maioria de sistemas utilizados atualmente em pesquisa, de forma integrada ao próprio experimento, nos quais não há domínio completo da tecnologia neles envolvida. Isto cria uma dificuldade enorme quando o pesquisador necessita adaptar aqueles subsistemas às demandas impostas pelas novas variantes do experimento, culminando invariavelmente na própria frustração do experimento. Além disso, adiciona-se o fato de neste caso tratar-se de equipamentos que superam a faixa de um milhão de dólares americanos, o que dificulta muito a aquisição de versões com novas funcionalidades ou mesmo adaptações (*upgrades*), que se situam com valores tipicamente de centenas de milhares de dólares. Junta-se também a enorme inércia imposta pelos fornecedores de tais instrumentos em atender a necessidades específicas dos pesquisadores em incorporar novas funcionalidades, basicamente imposta por decisões marcadas por regulações mercadológicas.

Com o surgimento das FPGAs, que possibilitam, ao mesmo tempo, realizar tarefas com a criticidade temporal necessária em RM, garantir a flexibilidade das partes que compõem o sistema e reduzir os efeitos da obsolescência, o CIERMag adotou, então, como a principal característica de *hardware* que compõe o espectrômetro, o uso da abordagem de *hardware* reconfigurável utilizando estes componentes. Esta decisão acarretou na necessidade da criação de diversos subsistemas, principalmente na parte de *software*, que pudessem contemplar toda a flexibilidade dos recursos desta tecnologia, tendo em vista que os sistemas desenvolvidos anteriormente pelo grupo faziam uso de tecnologias que, atualmente, estão defasadas e eram de caráter específico para o equipamento utilizado na época.

O advento das tecnologias de Lógicas Programáveis veio reverter uma condição estabelecida na década de 1990. Esta condição decorre principalmente da imposição pela necessidade de dutos de comunicação de dados cada vez mais rápidos, para que fossem compatíveis com a crescente velocidade de processamento dos processadores, e cuja engenharia de produto não encontrava solução satisfatória para que projetos de *hardware* continuassem a ser executados em ambiente de pesquisa, como era feito até então. Neste momento, outro conceito, o do *Personal Computer* (PC), ou Computador Pessoal, começou a ficar cada vez mais disponível e barato, o que resultou na migração para o desenvolvimento de *software* (aplicações) de um grande volume de recursos humanos que se dedicava ao desenvolvimento de ambos os conceitos, de *hardware* e *software*.

Assim, a possibilidade de desenvolver dispositivos baseados em Lógicas Programáveis não contemplava mais as limitações de Engenharia de Produto, e restaurou, pelo menos parcialmente até este momento, a oportunidade para que ambos os conceitos pudessem ser desenvolvidos em um mesmo projeto. A figura 9 mostra o espectrômetro do CIERMag, desenvolvido com o uso de FPGA, ligado ao osciloscópio para fins de testes.

Figura 9 - Espectrômetro digital do CIERMag ligado ao osciloscópio para realização de testes.

Fonte: Documentação interna do CIERMag.

Nesta seção os principais elementos e alguns conceitos que fazem parte da nova versão do espectrômetro do CIERMag serão abordados, com exceção dos que fazem parte deste trabalho de mestrado, que é o caso do *ToRM Console* e *ToRM Processing API*, que serão vistos nos capítulos 4 e 5. A leitura e entendimento destes elementos são essenciais para a compreensão deste projeto, tendo em vista o alto nível de integração entre todos eles.

3.2 Métodos de sequências de pulsos, Linguagem F e *ToRMC*

Como visto no capítulo 2, para observar sinais de ressonância magnética deve-se excitar o sistema de *spin*, de maneira que a magnetização saia da sua condição de equilíbrio e, em seguida, observar o comportamento do sistema através da leitura dos sinais eletromagnéticos gerados pelo sistema de núcleos atômicos. Para isso, a aplicação de pulsos de radiofrequência por meio de bobinas transmissoras é necessária. Esses pulsos são caracterizados por uma quantidade grande de parâmetros como, por exemplo, a duração, a potência utilizada, a largura de banda, entre outros. Além disso, normalmente um conjunto idêntico de pulsos é repetido diversas vezes, de maneira que apenas alguns parâmetros envolvidos podem ser variados entre uma repetição e outra.

Como método entende-se o “projeto” de um experimento, e compreende vários itens: um deles é a sequência de pulsos, que determina de que forma se comportarão gradientes, RF, aquisição de dados e loops de repetição; uma sequência de pulsos é sempre parametrizada para reutilização, e o que garante isso são parâmetros que, mediante controle do usuário, permite que uma sequência seja utilizada em *scans* com diferentes propósitos; uma terceira é a interface com o usuário, específica do método em questão e que permite estabelecer a inter-relação entre os parâmetros do método, acessíveis através de um *script* de validação. Um conjunto de valores correntes daqueles parâmetros constitui uma atribuição única de um *scan*, e pode ser armazenado como um *protocolo de execução*; por fim, cada método prevê um conjunto de ações de pós-processamento, caracterizado pelo *script* de processamento que permite fazer de forma automática e reproduzível todas as operações de processamento dos dados adquiridos e armazenados num formato padronizado, uma das características principais deste projeto. Algumas vezes esse termo é descrito como método, *sequência de pulsos* ou, simplesmente, *sequência*, por mera economia de argumentos.

A princípio, todas as formas de ondas e parâmetros poderiam ser armazenados em memória no espectrômetro. Entretanto, como a escala de tempo desses pulsos é da ordem de microssegundos e sabendo que o tempo de execução de um experimento de ressonância magnética é, em média, da ordem de alguns minutos, uma quantidade muito grande de memória seria necessária, inviabilizando esta abordagem. Além disso, caso algum parâmetro que define o formato de algum pulso fosse modificado, as formas de ondas deveriam ser novamente gravadas. Deste modo, aproveitando a existência de repetições desses conjuntos de pulsos e a utilização de processadores que possam executar as operações lógicas necessárias

para a execução de um experimento, códigos em baixo nível são enviados para o espectrômetro anteriormente a cada execução para que os processadores do espectrômetro gerem as formas de ondas, bem como as demais operações necessárias. Esta é uma abordagem utilizada neste projeto e que visa utilizar o conceito de “*compile on demand*”, ao contrário do convencional modelo de manter um módulo executável, que possa ser modificado mediante parâmetros (variáveis, ou argumentos) de entrada. Na versão atual do espectrômetro, é possível inclusive decidir quais as partes da sequência de pulsos que será compilada ou não.

Para permitir a programação em nível mais alto foram realizados pelo grupo a concepção e desenvolvimento da linguagem F e seu respectivo compilador, *ToRMC*. (22) O *ToRMC* é desenvolvido em C++ e até o momento possui versões que são utilizáveis no sistemas operacionais *Linux* e *Windows*. Do ponto de vista conceitual, o objetivo da linguagem F é fazer com que o programador de sequências de pulsos fique mais próximo da abordagem física do experimento. Sendo assim, idealmente, após a criação do projeto de um método, o programador estará apto a implementá-lo no espectrômetro do CIERMag. Em função da repetição de conjunto de pulsos, a linguagem F é utilizada principalmente para a criação de *loops*, que são iterações semelhantes aos das existentes em outras linguagens como no comando *for* contido na linguagem C, por exemplo.

Além disso, para que sejam reaproveitados os diversos trechos que podem ser comuns entre os diversos métodos, a estrutura do código da sequência de pulsos é dividida em sequência (*sequence*), módulos (*modules*) e eventos (*events*) conforme a figura abaixo:

Figura 10 - Estrutura do código de um método de sequência de pulsos escrito na linguagem F. Um mesmo método possui um único trecho de código denominado *sequence* (sequência) que pode conter um ou mais *modules* (módulos) que, por sua vez, podem conter um ou mais *events* (eventos). Módulos distintos podem fazer referência aos mesmos eventos, conforme indicado na figura. Além disso, dentro dos módulos estão contidas as definições dos *loops* que determinam a sequência de pulsos.

Fonte: Elaborada pelo autor.

Outra característica da linguagem F é permitir que tabelas sejam inicializadas a partir de arquivos que contenham um conjunto de valores que podem ser utilizados para definir formas de onda ou para determinar ordem de decodificação de fase, utilizada na aquisição de imagens, por exemplo. Estas tabelas também podem ser definidas dentro do código da sequência. Além disso, tabelas são fundamentalmente diferenciadas entre “*shapes*” – essencialmente formas de onda de pulsos de RF e de gradientes – e “*tables*” – todas as outras formas de tabelas, multiplicativas, de *offset* de frequências, de fases *etc.*

Todos os parâmetros que não possuem seus valores inicializados dentro do código da sequência devem ser passados através de um arquivo de texto, denominado protocolo de execução, que será discutido adiante. A localização deste arquivo deve ser informada como parâmetro de execução ao compilar um método. A figura abaixo mostra um trecho de código escrito na linguagem F:

```

29 module Balanced{
30     NumPhEnc, NumAver, NumSlices,
31     tSel, tRise, tFall, tRise1, tFall1, tPre, tPos, tPrep, tRead, tAcq, tTrig,
32     ExcAMamp, ExcPHamp, ACQPHamp,
33     SelGSelamp, PrepGSelamp, PrepGRdamp, RoutGRdamp,
34     TR
35 }:
36
37 start Pre_Selection_Balanced(tRise, tPre, SelGSelamp);
38 loop Slice = 0, NumSlices - 1, type = slices:
39     loop PhEnc = 0, NumPhEnc - 1, type = PhEl:
40         loop Average = 0, NumAver - 1, type = average:
41             rotation(PlaneOrientation[(Slice) % len(PlaneOrientation)]);
42             start ScopeTrigger(tTrig);
43             start Selection_Balanced(tSel, tFall, tPos, ExcAMamp, ExcPHamp, SelGSelamp, ExcFreqOff[Slice]);
44             start PrepSPR(tPrep, tRise, tFall, PrepGSelamp, Encoding[PhEnc], PrepGRdamp);
45             delay DL1 = (TR - tSel - tPrep - tRead - tPrep - tRise)/2;
46             start ReadoutR(tRead, tRise1, tFall1, ACQPHamp, RoutGRdamp, 1);
47             delay DL2 = DL1;
48
49             start PrepSPR(tPrep, tRise, tFall, PrepGSelamp, Encoding[NumPhEnc-PhEnc-1], PrepGRdamp);
50
51             start Pre_Selection_Balanced(tRise, tPre, SelGSelamp);
52         end
53     end
54 end
55 end

```

Figura 11 - Trecho do módulo ‘Balanced’ escrito em linguagem F. A inicialização do módulo ocorre entre as linhas 29 e 35, onde o mesmo recebe os parâmetros oriundos do protocolo de execução. As linhas 37, 42, 43, 44, 46, 49 e 51 são chamadas para eventos, que estão definidos em outros arquivos que devem ser importados no início do código. As linhas 38, 39 e 40 definem *loops*, ou iterações, de maneira que o primeiro parâmetro define o nome da variável atribuída ao índice do *loop* e o valor inicial do mesmo, o segundo parâmetro define o último valor do índice e o parâmetro *type* define o tipo de *loop*, que será discutido no capítulo 4. Os *loops* são finalizados nas linhas 52, 53 e 54. As linhas 45 e 47 são *delays*, ou atrasos, utilizados para que o espectrômetro aguarde um determinado intervalo de tempo até a execução da próxima instrução. A linha 55 indica o término do módulo ‘Balanced’.

Fonte: Elaborada pelo autor.

O produto final da execução do compilador é composto por diversos arquivos de propósitos distintos. O principal arquivo, cujo nome também deve ser informado como parâmetro de execução, contém os códigos, que são *opcodes* em base hexadecimal, que devem ser carregados anteriormente à execução de um método. Outro arquivo de saída contém informações a respeito da estrutura de *loops*, conforme será visto no capítulo 4. Outras informações referentes principalmente à depuração durante a etapa de desenvolvimento também são geradas em outros arquivos de saída após o processo de compilação.

3.3 *ToRM IDE*, Editor gráfico de sequências e Editor F

Embora a linguagem F possa ser utilizada com editores de texto comuns e o seu compilador, *ToRMC*, não necessite obrigatoriamente de nenhum outro recurso para a criação dos arquivos de saída, a maneira mais adequada de utilizar os recursos que permitem reaproveitamento de trechos de códigos, ou seja, o reaproveitamento de *events* e *modules*, deve ser feita através de ferramentas especializadas. Essa subseção tem como objetivo descrever algumas ferramentas desenvolvidas pelo próprio grupo para o auxílio do desenvolvimento de métodos de sequência de pulsos.

Duas abordagens distintas, porém equivalentes, coexistem no espectrômetro do CIERMag. A primeira delas consiste, de maneira tradicional, da criação dos códigos, escritos diretamente em linguagem F. Para esta abordagem, apenas os recursos básicos do *ToRM IDE*, em conjunto com o editor F, que serão discutidos a seguir, serão necessários. A segunda, embora não tão geral, permite que, de maneira gráfica, seja possível a criação de métodos com o auxílio das ferramentas de edição gráfica, que, apesar de constituir um subsistema à parte, deve ser disparado de dentro do contexto do *ToRM IDE*, embora não necessite do auxílio do editor F.

O principal papel do *ToRM IDE* é criar ferramentas gráficas de gerenciamento dos parâmetros, que podem ser pertencentes ao método ou ao próprio sistema, e suas respectivas formas de validação. Um dos seus principais elementos gráficos, a aba de parâmetros, será um dos recursos utilizados pelo *ToRM Console*, por questões de reaproveitamento de código e uniformização, conforme será discutido no capítulo 4. A figura abaixo mostra o *ToRM IDE* sendo executado:

Figura 12 - ToRM IDE em execução no gerenciamento de parâmetros do método 'CPMG_E'.

Fonte: Documentação interna do CIERMag.

O editor F é um *plug-in* do editor *Eclipse*, da *Eclipse Foundation*. (23) Desenvolvido em Java, este *plug-in* permite com que os recursos do *Eclipse*, um editor já bem estabelecido em diversos sistemas operacionais, possam ser utilizados para edição de códigos escritos na linguagem F, permitindo sua utilização como um “*language sensitive editor*”.

Entre as funcionalidades existentes, estão os recursos de realce de sintaxe, onde as palavras reservadas ficam destacadas e os erros de sintaxe são apontados em tempo de edição, aumentando a legibilidade do código e reduzindo o tempo com depuração. Além disso, é possível também navegar diretamente para a definição dos módulos, eventos, bem como utilizar sugestões para completar o código, conforme o mesmo está sendo digitado. A figura 13 mostra um trecho de código sendo editado de dentro do editor F.

O editor gráfico de sequencias, disparado de dentro do *ToRM IDE*, permite a criação de eventos, módulos e sequências utilizando somente recursos gráficos, conforme ilustrado na figura 14. Como produto final do editor, um código em linguagem F é criado automaticamente pelo editor gráfico, de maneira que o mesmo possa ser compilado posteriormente. O conjunto de parâmetros e validações devem ser, da mesma forma que a abordagem convencional, tratados pelo *ToRM IDE*.

A figura 14 mostra o editor gráfico de sequências em execução.

Figura 13 - Eclipse com o *plugin* do editor F em execução durante a edição de uma biblioteca de eventos.

Fonte: Documentação interna do CIERMag

Uma abordagem para a transcrição do código “F” para sua representação gráfica está sendo finalizada neste momento (Jan/2014), e deverá completar as possibilidades de edição e translação de quaisquer formas de desenvolvimento (gráfico → *line programming* → gráfico).

Figura 14: Editor gráfico de sequências sendo executado dentro do escopo do *ToRM IDE*. O módulo ‘*SE_Multislice*’ é mostrado.

Fonte: Documentação interna do CIERMag.

3.4 ACQServer

Conforme será visto no capítulo 4, a arquitetura escolhida para o sistema de aquisição é baseada na arquitetura cliente-servidor, por trazer algumas vantagens que serão discutidas posteriormente. Sendo assim, o *ACQServer* é a aplicação desenvolvida pelo grupo que possui o papel de se comunicar com o *hardware* que compõe o espectrômetro e, além disso, prover uma interface de comunicação com o sistema de aquisição do *ToRM Console*.

A comunicação entre o espectrômetro e o *ACQServer* pode ser feita através do padrão de comunicação *Universal Serial Bus* (USB), que possui taxa de transferência de 480 Mbps, ou do protocolo *Gigabit Ethernet* (GbE). Ao executar o *ACQServer*, o protocolo de comunicação mais rápido é escolhido automaticamente.

A interface de comunicação entre o sistema de aquisição do *ToRM Console* e o *ACQServer* é feita pelo protocolo *File Transfer Protocol* (FTP) ligeiramente adaptado. As adaptações no protocolo foram feitas para adicionar novos comandos que possibilitam o carregamento de métodos e parâmetros de sistema, além de envio de comandos diretamente para o espectrômetro. A transferência dos dados é feita através de arquivos no formato *HDF5*, que será discutido com mais detalhes no capítulo 4. (24) A figura abaixo mostra o *ACQServer* em funcionamento.

Figura 15 - *ACQServer* em funcionamento durante a carga do método 'CPMG_E'. Os comandos recebidos foram enviados pelo *ToRM Console*, conforme será discutido no capítulo 4.

Fonte: Elaborada pelo autor.

4 TORM CONSOLE

4.1 Introdução

Este capítulo aborda a etapa de concepção e desenvolvimento do subsistema responsável pela execução de métodos de sequências de pulsos, previamente desenvolvidos com o auxílio das ferramentas mencionadas no capítulo 3. O *ToRM Console* tem como principal objetivo fazer com que usuários com diferentes níveis de conhecimento em relação à técnica utilizada ou ao equipamento possam operar o espectrômetro utilizando os métodos existentes, de maneira prevista pelo programador do mesmo.

Para que seja possível a execução de um experimento, denominado *scan*, várias etapas devem ser percorridas. Entre elas estão: a preparação do método de acordo com as necessidades do operador; e a obtenção, filtragem, organização, visualização, armazenamento e pós-processamento dos dados. Deste modo, o *ToRM Console* tem como fim atender aos seguintes requisitos:

1. Fornecer um sistema de múltiplos usuários;
2. Listar todos os métodos disponíveis ao usuário autenticado;
3. Preparar método;
 - a. Após o carregamento do método, permitir com que o usuário faça alterações nos parâmetros disponíveis;
 - b. Validar parâmetros por meio de execução de *script* de validação e garantir que seus respectivos valores estejam dentro do intervalo permitido;
 - c. Compilar método.
4. Executar método;
 - a. Estabelecer comunicação com o espectrômetro por meio do *ACQServer*;
 - b. Enviar método compilado ao servidor e carregá-lo no *hardware* através de comandos apropriados;
 - c. Inicializar os valores dos *parâmetros de sistema* e de *tempo real* do protocolo de execução utilizado;
 - d. Inicializar *scan*, levando em consideração o módulo escolhido para ser executado.

5. Realizar aquisição dos dados;
 - a. Obter os arquivos gerados pelo *ACQServer*, levando em consideração a localização do servidor, que pode ser local ou remota;
 - b. Aplicar filtros passa-baixa do tipo FIR, de acordo com a banda informada no protocolo de aquisição;
 - c. Visualizar o último dado filtrado, em tempo quase real^{vii};
 - d. Organizar dados filtrados de acordo com o protocolo de aquisição;
 - e. Armazenar os dados filtrados e organizados, bem como os protocolos de execução e aquisição no diretório de trabalho do usuário;
6. Realizar etapa de pós-processamento de acordo com *script* de processamento^{viii}.
7. Permitir com que seja possível o envio de comandos para o espectrômetro através de linha de comando.

Todo o desenvolvimento desta aplicação foi realizado em linguagem de programação *Python*. Esta abordagem permite com que o *ToRM Console* possa ser utilizado em todos os sistemas compatíveis com os interpretadores e bibliotecas utilizados dessa linguagem. Durante a etapa de realização deste projeto os sistemas operacionais *Windows*, nas versões 7 e 8, bem como as distribuições *Debian* e *Ubuntu* do *Linux* foram utilizados, de modo alternado, de maneira a garantir a portabilidade entre eles. (25-26) Outros sistemas *desktop*, como o *Mac OS X*, por exemplo, poderiam ser utilizados, embora nenhum teste tenha sido feito por não fazerem, atualmente, parte do escopo do projeto.

4.2 Interface Gráfica

Por se tratar de um *software* para sistemas *desktop*, a interface gráfica com o usuário possui um papel fundamental do ponto de vista da usabilidade. Conforme mencionado no capítulo 1, uma variedade de *frameworks*, ou arcabouços, para a criação de interfaces gráficas pode ser encontrada na forma de bibliotecas disponíveis no *Python*. Entretanto, as bibliotecas *PySide* e *PyQt*, que realizam o encapsulamento do *Qt framework*, originalmente escrito em

^{vii} O tempo necessário para filtragem varia de acordo com o número de pontos obtidos e, portanto, atrasos na etapa de visualização podem ser observados em alguns casos.

^{viii} Este item será abordado no capítulo 5, durante a discussão do *ToRM Processing API*.

linguagem C++, foram as principais candidatas a fazer parte do conjunto de tecnologias utilizadas neste trabalho. Ambas as bibliotecas atendem aos requisitos deste trabalho, além de possuírem sintaxe muito semelhante. No entanto, a biblioteca *PySide* foi escolhida por motivos relacionados ao tipo de licença da mesma (LGPL), que não restringe a maneira de distribuição e comercialização deste *software*.

O principal objetivo da biblioteca *PySide* é encapsular o *Qt*, ou seja, apenas fazer com que seja possível utilizá-lo em linguagem *Python*. Deste modo, tanto a documentação, quanto a comunidade virtual de usuários de ambas as ferramentas podem ser consultadas durante a etapa de desenvolvimento, levando em consideração o fato que a biblioteca *PySide* mantém a mesma nomenclatura para os métodos e classes do *Qt*. Por este motivo, a distinção entre uma ferramenta e outra durante a explicação dos elementos gráficos utilizados não será necessária.

Duas formas podem ser utilizadas para a criação de janelas no *PySide*. A primeira delas é realizada através de programação, onde se devem instanciar objetos das classes dos elementos gráficos desejados, criar as relações hierárquicas entre esses elementos e, por fim, utilizar os métodos adequados para exibição. Outra maneira se dá através do *Qt Designer*, um *software* que permite a criação de janelas através de uma ferramenta gráfica. (27) Os códigos gerados pelo *Qt Designer* podem ser convertidos para linguagem *Python* através da ferramenta *PySide-UIC*, de modo a acelerar o processo de desenvolvimento das interfaces gráficas. O código gerado automaticamente não deve ser modificado, uma vez que possíveis alterações na interface gráfica poderão ser feitas pelo *Qt Designer*. Um arquivo auxiliar deve ser utilizado para realizar a importação do código gerado automaticamente, bem como incorporar as alterações e implementações que não podem ser feitas dentro do *Qt Designer*. As duas abordagens foram utilizadas ao longo deste trabalho de acordo com a complexidade de cada uma das janelas desenvolvidas.

4.3 Estrutura dos diretórios do sistema

Os diversos subsistemas que compõem o espectrômetro do CIERMag, discutidos no capítulo 3, possuem como o maior fator de integração o compartilhamento das mesmas definições dos diretórios do sistema. Estes diretórios possuem subdiretórios e arquivos que contém informações acerca dos métodos disponíveis pelo sistema, configurações, bibliotecas escritas em linguagem F e os arquivos executáveis das demais aplicações como, por exemplo,

o compilador *ToRMC* e o próprio *ToRM Console*. Nesta subseção será apresentada a estrutura dos diretórios do sistema. A figura abaixo mostra todos os subdiretórios contidos no diretório do sistema:

Figura 16 - Hierarquia do diretório do sistema. Nesta figura estão contidos apenas os subdiretórios que o compõe, omitindo-se os arquivos.

Fonte: Elaborada pelo autor.

No primeiro nível da hierarquia mostrada na figura 16 (vermelho) estão as pastas ‘bin’, ‘tmp’, ‘sys’ e ‘usr’. A pasta ‘bin’ contém todos os executáveis dos subsistemas do espectrômetro do CIERMag. Em sua pasta raiz está contido o executável do compilador *ToRMC* que deve possuir o nome ‘*tormcwin32.exe*’ quando executado nos sistemas operacionais *Microsoft Windows* ou ‘*tormclinux*’ quando executado no *Linux*. Os demais subsistemas estão contidos em seus respectivos subdiretórios. A pasta ‘tmp’ serve para guardar qualquer arquivo de caráter temporário utilizado pelo sistema. Em especial, o subdiretório ‘*ACQServerHome*’ (cinza) é utilizado para armazenar os arquivos gerados pelo servidor *ACQServer*. A pasta ‘usr’ era, inicialmente, utilizada para armazenar todos os métodos dos diversos usuários. No entanto, com o uso de um diretório pessoal chamado ‘*CIERMag Workingplace*’ para este fim, a ser discutido a seguir, o diretório ‘usr’ contém, atualmente, apenas os arquivos de configuração do *ToRM IDE* e um arquivo de configuração, ‘*usr_cfg.json*’, utilizado pelo *ToRM Console* para indicar onde está contido o diretório pessoal de cada usuário.

No subdiretório *'sys'*, estão contidos os subdiretórios *'api'*, *'lib'* e *'meth'* (rosa). A pasta *'api'* contém os arquivos utilizados pelo *ToRM IDE* e pela aba de parâmetros do *ToRM Console*, também herdada do *ToRM IDE*. Estes arquivos constituem uma biblioteca *Python* para manipulação dos parâmetros de sistema. O subdiretório *'lib'* contém valores tabelados e códigos, escritos em linguagem F, que podem ser reaproveitados durante o desenvolvimento de novos métodos, que devem ficar no interior dos subdiretórios *'events'*, *'modules'* e *'shapes'* (roxo).

Por fim, o diretório *'meth'* contém métodos chamados de *métodos do sistema*, que são disponíveis para todos os usuários do espectrômetro. No interior desta pasta devem conter subdiretórios nomeados de acordo com o nome do método, que contém todos os arquivos que os compõem. Estes arquivos devem ficar no interior dos diretórios *'source'*, *'build'* e *'protocols'* (laranja). A pasta *'source'* contém os principais arquivos que constituem o método do sistema, ou seja; o código escrito em linguagem F da sequência de pulsos, os arquivos contendo os parâmetros do método e seu respectivo *script* de validação. A pasta *'build'* é utilizada para armazenar, temporariamente, os arquivos gerados pelo compilador. Os protocolos de execução e aquisição, que serão detalhadamente descritos respectivamente nas subseções 4.5 e 4.8, e o *script* de validação são armazenados nos subdiretórios *'acquisition'*, *'execution'* e *'processing'* (azul). Todos os arquivos de configuração do sistema devem ser protegidos pelo sistema operacional, de tal modo que apenas usuários com os privilégios administrativos consigam alterá-los por motivos de segurança.

O diretório *'CIERMag Workingplace'* assemelha-se ao diretório do sistema, discutido acima, conforme mostrado na figura 17. No entanto, além de não possuir alguns subdiretórios como *'sys'* e *'bin'*, os diretórios *'meth'* e *'lib'* estão contidos em hierarquia mais alta. O principal papel do *'CIERMag Workingplace'* é manter, de forma estruturada, os dados, métodos e outros arquivos particulares do usuário. Sendo assim, no mesmo computador podem existir vários desses diretórios, que devem ser colocados em locais com restrições de acesso. O diretório *'data'* é utilizado para manter todos os dados adquiridos pelo usuário correspondente do espectrômetro.

Figura 17 - *CIERMag Workplace*. Este diretório, semelhante ao diretório raiz do sistema, é de uso particular de cada usuário.

Fonte: Elaborada pelo autor.

4.4 Sistema de usuários

O uso de equipamentos de RM por parte de diversos usuários é uma prática muito comum. Portanto, é necessário gerenciar as permissões de acesso e prover diretórios particulares, onde os métodos e dados pertencentes aos distintos usuários devem ser armazenados. Os diretórios privados dos usuários devem ter suas permissões garantidas com o auxílio do sistema operacional utilizado. Deste modo, não é papel do *ToRM Console* garantir que as permissões de escrita e leitura dos arquivos contidos nestes diretórios sejam de acordo com as desejadas. Estas permissões devem ser gerenciadas pelos usuários que possuam acesso privilegiado no computador onde o *ToRM Console* será executado. Entretanto, é tarefa do *ToRM Console* identificar o usuário que está executando a aplicação, através da solicitação de nome de usuário e senha e, por meio do arquivo de configuração, *usr_cfg.json*, contido na pasta '*usr*' identificar o diretório '*CIERMag Workplace*' pertencente a este usuário.

Sendo assim, a primeira tela a ser apresentada após a execução do *ToRM Console* é mostrada na figura 18:

Figura 18 - Tela inicial (*System Login*) do *ToRM Console*. Os usuários pré-configurados no arquivo '*usr_cfg.json*' são listados no campo '*User*' e a respectiva senha deve ser digitada no campo '*Password*'. Os campos '*Database Host*' e '*ACQServer Host*' informam, respectivamente, o endereço da base de dados referente ao sistema de usuários e o endereço onde o servidor de aquisições, *ACQServer*, está localizado.

Fonte: Elaborada pelo autor.

A conexão com a base de dados é realizada através da biblioteca MySQLdb. (28) O esquema da base de dados é mostrado na figura 19. Apenas quatro tabelas são utilizadas para compor o sistema: '*users*', '*user_groups*', '*groups_permissions*' e '*permissions*'. A tabela '*users*' guarda todas as informações pessoais e de contato do usuário, além de informar o grupo que o mesmo pertence e, caso haja, o grupo temporário, conforme o sistema de permissões temporárias discutido a seguir. A senha do usuário (*tx_password*) é criptografada. A tabela '*user_groups*' define os grupos de usuários que possuem um conjunto de permissões, definidos pela tabela '*groups_permissions*'. Por sua vez, a tabela '*permissions*' contém todas as permissões para operações que possuem restrições. Por exemplo, para poder criar novos usuários é necessário possuir a permissão '*account_admin*'.

Figura 19 - Diagrama de relacionamento das tabelas que constituem o sistema de usuários.

Fonte: Elaborada pelo autor.

Quando o usuário pressiona o botão *'Logon'* (figura 18), o banco de dados é consultado, de maneira a verificar a existência do usuário e a validade da senha, emitindo mensagem de erro caso alguma irregularidade ocorra nesta etapa. Em seguida, é observado se o usuário possui a permissão *'login'*, que é necessária para a verificação de casos onde os usuários estejam bloqueados temporariamente por motivos administrativos. Logo após, um objeto da classe *Validation* é instanciado. A instância desta classe é necessária para futuras consultas de permissões, acesso a informações pessoais e outras operações relacionadas ao sistema de usuário.

Após o acesso ao sistema, é possível que o usuário consulte e altere suas informações pessoais, de acordo com a figura 20. A tela de gerenciamento de contas pode ser acessada através do menu *'System'*, na tela principal do *ToRM Console*, escolhendo a opção *'Account Management'*.

A segunda aba mostrada na figura 20, *'Accounts Administration'*, é responsável pelo gerenciamento de todas as contas de usuários do sistema. Ela somente é visível caso o usuário tenha a permissão *'account_admin'*. As seguintes operações administrativas são disponíveis: criação de novo usuário (*Add User*), deleção de usuário (*Delete User*), troca rápida de senha (*Change Password*) e alteração de informação do usuário (*Edit User*). As operações de alteração, deleção e troca de senha necessitam que um usuário seja selecionado na lista, gerando mensagem de erro, caso contrário. A figura 21 mostra o conteúdo dessa aba.

Figura 20 - Gerenciamento de conta. Visualização e alteração de dados pessoais.

Fonte: Elaborada pelo autor.

Figura 21 - Gerenciamento de contas. Administração de contas do sistema.

Fonte: Elaborada pelo autor.

A janela de criação de usuários é mostrada na figura abaixo. Dois grupos de informações devem ser preenchidos: '*Personal Information*', contendo as informações pessoais e '*Account Information*', contendo os dados de acesso e o tipo de usuário a ser cadastrado. Esta mesma janela foi reaproveitada para constituir a janela de alteração de dados.

The screenshot shows the 'ToRM Console - New user' window. It is divided into two main sections: 'Personal information' and 'Account information'. The 'Personal information' section includes fields for Name, E-mail, Address, Birth Date (with a dropdown menu), Telephone, and Mobile Phone. The 'Account information' section includes fields for User Level (with a dropdown menu), Username, and Password. At the bottom right, there are two buttons: 'Add User' and 'Clear'.

Figura 22 - Gerenciamento de conta. Criação de novos usuários.

Fonte: Elaborada pelo autor.

Os tipos de usuários definidos atualmente são:

- *Root*: possui todas as permissões do sistema.
- *User*: usuário comum. Possui apenas as permissões de acesso e operação do sistema.
- *Specialist*: usuário especialista. Usuário intermediário entre '*User*' e '*Root*'.
- *Inactive*: usuário inativo. Esse grupo pode ser utilizado quando é necessário manter as informações de um usuário existente, mas, por algum motivo, o usuário deve ser bloqueado temporariamente.

As permissões dos grupos podem ser acessadas através do submenu '*Permissions Management*', do menu principal '*System*'. Para acessar esta janela é necessário possuir a permissão '*permission_admin*'. A figura abaixo mostra a janela de administração das permissões dos grupos:

Figura 23 - Gerenciamento de permissões do *ToRM Console*. Nesta janela é possível controlar as permissões disponíveis para cada um dos grupos, caso o usuário autenticado possua permissão para tal tarefa.

Fonte: Elaborada pelo autor.

Outro recurso do sistema de usuário do *ToRM Console* é permitir a alteração temporária de grupo de um determinado usuário. Em alguns casos essa funcionalidade pode ser necessária para realizar alguma operação que requer permissão durante a execução de um experimento de longa duração, por exemplo. Portanto, é desejável que o experimento não seja interrompido e essa operação possa ser executada com a autorização de algum outro usuário que possa conceder tal permissão. Após a finalização da sessão, a alteração temporária de grupo é automaticamente desfeita. Para solicitar a alteração temporária de grupo o usuário deve acessar o submenu '*Grant Temporary Permission*', do menu principal '*System*'. A

figura abaixo mostra a requisição sendo feita para que o usuário atual receba, temporariamente, todas as permissões do grupo *Manager*:

Figura 24 - Tela de requisição de permissão temporária do *ToRM Console*. É possível que um usuário pertencente a um grupo de hierarquia maior possa conceder permissão temporária ao operador corrente.

Fonte: Elaborada pelo autor.

4.5 Preparação de *scan*

Depois da etapa de autenticação, todos os métodos disponíveis ao usuário estarão listados na primeira tela do *ToRM Console*, conforme mostrado na figura 25. O usuário deve, então, escolher um desses métodos com um clique duplo sobre o item da árvore ou pressionar a tecla *ENTER* ou *RETURN* no item selecionado para que o mesmo seja carregado.

Durante a carga do método, algumas etapas são percorridas. Entre elas, a mais relevante é de inicialização da aba de parâmetros. Como mencionado no capítulo 3, a aba de parâmetros é proveniente do *ToRM IDE*, o editor de sequências e gerenciador de parâmetros. O reaproveitamento de código foi possível devido ao fato de que tanto o *ToRM IDE*, quanto o *ToRM Console* são escritos na mesma linguagem que (*Python*). Para a compreensão da etapa de inicialização da aba de parâmetros os diversos elementos que compõem o método devem ser apresentados. Desta forma, esses elementos serão discutidos na mesma ordem em que eles aparecem durante esta etapa. Embora a estrutura de diretórios seja definida, conforme mostrado anteriormente, nenhuma definição é feita diretamente no código, de maneira que

todos os diretórios são obtidos de arquivos de configuração do método, do sistema e da própria aplicação. Sendo assim, durante o carregamento de um método todos os diretórios são informados à aba de parâmetros.

Figura 25 - Tela inicial do *ToRM Console* após etapa de autenticação. Na árvore de itens '*Methodologies*' estão contidas todos os métodos disponíveis ao usuário. Os métodos que estão sob o item '*System Methodologies*' estão disponíveis a todos os usuários, enquanto os que estão sob o item, neste caso, '*danilo*' são pertencentes exclusivamente a este usuário.

Fonte: Elaborada pelo autor.

A primeira informação a ser fornecida é a localização do diretório raiz do método, que pode estar contido no subdiretório '*meth*' do '*CIERMag Workingplace*' ou do diretório de sistema '*sys*', dependendo do método escolhido pelo usuário. Em seguida, a localização das bibliotecas do sistema e de usuário, dos parâmetros do sistema e seu respectivo *script* de validação.

O arquivo de parâmetros de sistema contém principalmente informações que estão diretamente ligadas ao *hardware* do sistema. Além disso, são armazenadas informações relevantes à aba de parâmetros como, por exemplo, o posicionamento, a visibilidade, o intervalo de valores permitidos, *etc.* O formato de armazenamento utilizado é um arquivo de texto contendo estrutura *JavaScript Object Notation* (JSON). (29) Inicialmente, este formato foi bastante utilizado para o intercâmbio de dados em aplicações *web*, mas pelo seu poder de representação de hierarquias e simplicidade ele pode ser utilizado para substituir o formato *eXtensible Markup Language* (XML), que possui excesso de informação em sua

representação. (30) Além disso, a existência de uma biblioteca padrão do *Python* (*json*) permite a manipulação deste formato com o mapeamento direto nos tipos nativos adequados da linguagem.

O *script* de validação dos parâmetros de sistemas é um arquivo contendo código *Python* utilizado para verificar se o conjunto destes parâmetros obedece às restrições de inter-relações entre eles. Ou seja, mesmo que o usuário seja impedido de colocar valores que desrespeitam os intervalos permitidos pelo próprio arquivo de parâmetros do sistema, é possível existir uma combinação indesejada de valores, ainda que, individualmente, eles não apresentem irregularidades. Sendo assim, o *script* de validação dos parâmetros do sistema tem como função impedir essa situação. Este *script* deve ser desenvolvido pelo programador do método de sequência de pulsos.

A próxima informação a ser enviada à aba de parâmetros é a localização dos parâmetros do método e seu respectivo *script* de validação. Este arquivo é análogo ao arquivo de parâmetros do sistema. Entretanto, apenas as informações concernentes ao método devem constituir estes parâmetros. Desta forma, podem existir diversos arquivos de parâmetros de método, um para cada um dos métodos existentes, enquanto que apenas um arquivo de parâmetro de sistema deve existir. O *script* de validação funciona de maneira análoga ao de sistema e também deve existir para cada método criado.

O último passo consiste no carregamento do protocolo de execução padrão, que também é um arquivo de texto contendo informações no formato JSON. Enquanto que os arquivos de parâmetros de sistema e método consistem na definição e configuração de todos os parâmetros, conforme mencionado, o protocolo de execução é responsável por armazenar um conjunto possível de valores destes parâmetros. Isto é, um protocolo de execução contém um conjunto de valores para os parâmetros de método e sistema de um determinado método de sequência de pulsos que pode ser utilizado para reproduzir um mesmo experimento.

Um protocolo de execução pode ser salvo em qualquer localização, embora seja recomendado mantê-lo no '*CIERMag Workingplace*' no subdiretório '*execution*', abaixo de '*protocols*'. No entanto, ao realizar o carregamento do método apenas o protocolo de execução padrão será carregado automaticamente. Este protocolo fica no subdiretório '*execution*' e recebe o nome de '*default.json*'. Após o carregamento, o usuário, através da aba de parâmetros, poderá carregar outro protocolo de execução manualmente.

Durante a etapa de carregamento do método o usuário é informado, através de uma barra de progresso, sobre o *status* da operação. A janela denominada '*Logger*', contida inicialmente no canto inferior direito da tela, mantém o registro do carregamento e mostra

mensagens de erro, caso necessário. A figura 26 mostra o novo contexto do *ToRM Console* após esta etapa.

Figura 26 - *ToRM Console* após carregamento do método *SE_Mul*. O sistema de visualização, 'Viewport', fica em espera e a aba de parâmetros permite a alteração dos valores dos parâmetros.

Fonte: Elaborada pelo autor.

A aba de parâmetros permite, durante a etapa de preparação de *scan*, realizar a alteração dos valores dos parâmetros, levando em consideração o intervalo de valores permitido. Sendo assim, não serão permitidas as entradas errôneas de valores, embora os erros de inter-relação possam existir.

Além da alteração dos valores dos parâmetros, durante a etapa de preparação do *scan* o usuário também deve escolher qual módulo deve ser utilizado, caso exista mais de um. Isto pode ser feito alterando o módulo selecionado no submenu 'Modules' contido no menu principal 'Methodology'. Por padrão, o primeiro módulo instanciado no código da sequência é selecionado automaticamente.

Conforme é possível observar na figura 26, o botão ‘*Start*’, que pode ser utilizado através da tecla de atalho *F9*, foi habilitado. Ao pressioná-lo o sistema de aquisição, a ser discutido na seção 4.7, é inicializado.

4.6 Validação e compilação

Após o usuário ter solicitado o início de um *scan*, antes de inicializar o sistema de aquisição o sistema passa por uma etapa de validação. A etapa de validação é constituída por duas partes: a execução do *script* de validação e a verificação dos valores dos parâmetros, de maneira a observar se os mesmos estão dentro do intervalo permitido. A execução do *script* de validação pode automaticamente fazer as devidas correções nos problemas causados pelas inter-relações dos parâmetros ou enviar uma mensagem de erro, abortando a execução. A escolha entre uma abordagem e outra depende diretamente do programador do método e deve ser feita com cuidado, tendo em vista o impacto desta decisão^{ix}.

A segunda parte da validação, que é a realização da verificação dos valores dos parâmetros de acordo com os intervalos definidos nos arquivos de parâmetros de sistema e parâmetros do método, é necessária em função da possibilidade do próprio *script* de validação levar o conjunto de valores dos parâmetros a um estado inapropriado. Ou seja, caso a abordagem de correção automática de valores seja utilizada ou o *script* de validação seja usado para o cálculo do valor de algum parâmetro^x os valores máximos e mínimos dos parâmetros devem ser validados. Um protocolo de execução é gerado automaticamente após o processo de validação.

Em seguida, é dado início ao processo de compilação através de chamada ao *ToRMC*. O compilador recebe os seguintes parâmetros de entrada:

- Localização do código da sequência de pulsos, escrito em linguagem F (Obrigatório);
- Localização do diretório de bibliotecas do sistema (Opcional);
- Localização do diretório de bibliotecas do usuário (Opcional);

^{ix} A correção automática dos problemas pode acarretar na execução de um experimento indesejado por parte do operador. Por outro lado, o envio excessivo de mensagens de erro e a necessidade de correção manual pode aumentar significativamente o tempo de preparo de um experimento.

^x Alguns parâmetros ocultos e que são necessários para o método podem ser calculados através dos valores de outros parâmetros disponíveis ao usuário dentro do *script* de validação.

- Localização de outras bibliotecas utilizadas (Opcional);
- Localização do protocolo de execução, contendo os valores de todos os parâmetros utilizados (Obrigatório);
- Localização do arquivo de saída, em código hexadecimal (Obrigatório);
- Ativação do uso de *'hardpulses'*^{xi} (Opcional);
- Ativação de geração de arquivos para depuração (Opcional).

Todas essas informações estão contidas em um arquivo que define as dependências e configurações do projeto do método. Este arquivo deve possuir o mesmo nome do método, com formato *ToRM Project (trp)*^{xii}.

Como resultado da compilação são gerados os arquivos contendo as instruções, em hexadecimal, para o espectrômetro, os arquivos de depuração (caso seja solicitado), o protocolo de aquisição, que será discutido detalhadamente na seção 4.8 e o arquivo de variáveis de tempo real. Uma variável de tempo real é um tipo especial de parâmetro que pode ser alterado durante a execução de um experimento. Como a posição de memória que é utilizada para a leitura e escrita de valores de um parâmetro de tempo real é determinada apenas em tempo de compilação, o arquivo de variáveis de tempo real, *'RTVar'*, é necessário para mapear estas posições através de seus respectivos nomes.

4.7 Sistema de aquisição

Após a etapa de validação e compilação, caso o resultado tenha sido positivo, o sistema de aquisição é iniciado. Conforme mencionado no capítulo 3, o sistema possui uma arquitetura cliente-servidor, de maneira que os subsistemas *ToRM Console* e *ACQServer* não são obrigatoriamente disparados no mesmo computador, embora isso possa acontecer. A figura 27 mostra a interação entre os subsistemas do espectrômetro do CIERMag, levando em consideração a existência da arquitetura cliente-servidor.

^{xi} *Hardpulses* são pulsos que contêm formas de onda que necessitam de variações de tensões muito abruptas durante a aplicação dos mesmos. A técnica utilizada para permitir o uso deste tipo de pulso necessita que o compilador seja informado, de maneira que o código a ser carregado no espectrômetro possua as informações corretas para que o interpolador não suavize este tipo de forma de onda.

^{xii} O formato *'trp'* foi definido por convenção, mas possui o mesmo formato dos demais arquivos de configuração apresentados anteriormente. Isto é, ele é um arquivo de texto com informações contidas no formato JSON.

Figura 27 - Interação entre os subsistemas do espectrômetro do CIERMag, levando em consideração a arquitetura cliente-servidor utilizada. A edição, compilação e o controle e preparação do *scan* são feitos no lado do cliente, enquanto que a comunicação direta com o *hardware* é feita através do servidor. O *ToRM Console* (cliente) e o *ACQServer* (servidor) constituem a ponte de comunicação entre os dois lados.

Fonte: Elaborada pelo autor.

A comunicação entre o *ACQServer* e o *ToRM Console*, conforme já mencionado no capítulo 3, é realizada através do protocolo FTP, com a inclusão de comandos relacionados à carga e execução de um método de sequência de pulsos e comunicação direta com o *hardware*. Sendo assim, durante a inicialização do sistema de aquisição um conjunto de informações acerca da preparação do *scan* de um determinado método deve ser enviado ao *ACQServer* e, em seguida, o *ToRM Console* deve aguardar os dados que serão enviados pelo servidor.

O primeiro passo para início da execução de um experimento é, então, estabelecer uma conexão entre o *ToRM Console* e o *ACQServer*. Esta comunicação é feita através de uma biblioteca padrão do *Python*: *ftplib*. (31) Todas as operações que envolvem a comunicação com o *ACQServer* são implementadas na classe *SpecClient*, que faz uso da *ftplib*. Após o estabelecimento da conexão são enviados os comandos *'EXEC S'* e *'EXEC RST'* ao *ACQServer*, de maneira a interromper uma possível execução anterior e reiniciar o estado do espectrômetro. Além disso, o método *clear_files* da classe *SpecClient* é executado para que todos os arquivos contidos no diretório temporário do *ACQServer*, *'ACQServerHome'*, sejam apagados.

Em seguida, dois dos arquivos gerados durante o procedimento de compilação devem ser enviados ao servidor: o código hexadecimal da sequência da compilada; e o arquivo de variáveis de tempo real. O envio é feito através do método *send_file* da classe *SpecClient*, que deve receber como parâmetro uma instância do arquivo aberto em modo de leitura. O

carregamento destes arquivos é realizado através dos comandos *SEQ* e *RTVAR* do *ACQServer*, que são enviados através dos métodos *load_sequence* e *load_real_time_vars*, respectivamente, da *SpecClient*. Sendo assim, após o envio destes comandos, as instruções para execução da sequência de pulsos estarão carregadas no *hardware* do espectrômetro e o *ACQServer* terá o mapeamento de todos os parâmetros de tempo real, de maneira a permitir a leitura e alteração dos valores destes parâmetros através dos comandos *GET* e *SET*.

A próxima etapa consiste no carregamento de parâmetros do sistema. Os parâmetros do sistema podem ser obtidos através do comando ‘*SYS*’ do *ACQServer*. Este comando é implementado através do método *get_sys_parameters* da *SpecClient*, que retorna uma lista contendo os nomes dos parâmetros e os tipos de permissões de escrita e leitura. Um parâmetro do sistema pode ser apenas de leitura (‘*r*’), apenas de escrita (‘*w*’) ou de leitura e escrita (‘*rw*’). A partir do método *systemParametersValues* da aba de parâmetros, é possível receber todos os valores dos parâmetros de sistema escolhidos pelo usuário na etapa de preparação do *scan*. Sendo assim, todos os parâmetros que possuem permissão de escrita, ‘*w*’ e ‘*rw*’, são carregados no espectrômetro através do comando *SET* do *ACQServer*. De maneira semelhante, todos os parâmetros de tempo real são carregados. A varredura dos valores iniciais destes parâmetros também é feita através do protocolo de execução. Todos os parâmetros de tempo real são carregados através do método *update_real_time* da classe *SpecClient*.

Uma pasta contendo o nome do método de sequência de pulsos em execução é criada dentro do diretório ‘*data*’ do ‘*CIERMag Workingplace*’. Este diretório é utilizado para armazenar os dados que serão adquiridos, bem como as demais informações necessárias acerca do método corrente, conforme será discutido na seção 5.3.

A partir deste estágio, o *ToRM Console* passa a executar através de três processos concorrentes referentes à aquisição de dados, filtragem de dados e visualização. Esses processos são criados com o auxílio da biblioteca *Multiprocessing*, que também é uma biblioteca padrão do *Python*. (32) De fato, apenas os processos de aquisição e filtragem são criados, uma vez que a visualização é realizada pelo processo principal, que já está em execução.

O sincronismo destes processos é realizado através da classe *ExperimentController*, que tem o papel de acompanhar o andamento destes processos. A estratégia utilizada na implementação desta classe é baseada na solução do *problema dos produtores e consumidores*. (33) Duas relações de produtor e consumidor são tratadas simultaneamente pela classe *ExperimentController*: o processo de aquisição representa o produtor em relação

ao processo de filtragem (consumidor); enquanto que o processo de visualização é consumidor do processo de filtragem, desta vez, produtor.

A execução poderá ser interrompida a qualquer momento pelo usuário através do botão ‘*Stop*’ ou por algum problema encontrado em algum processo. Para que isso seja possível, é utilizada a técnica de programação concorrente denominada ‘*Poison Pill*’ ou ‘Pílula de Veneno’, que consiste em manter uma variável compartilhada para sinalizar a parada, que deve ser observada por cada um dos processos.

Como os processos podem escrever e ler simultaneamente sobre esta mesma variável, é possível que exista problemas de *condição de corrida*, que é um problema típico encontrado em programação concorrente. Entretanto, a biblioteca *Multiprocessing*, através de um tipo especial denominado *Value*, que, na verdade, retorna um tipo de dado compatível com os tipos da linguagem C (*ctypes*), permite com sejam utilizados *Locks*, ou travas, de modo a garantir que apenas um processo realize o procedimento de escrita ou leitura por vez. Uma instância de *Value*, utilizando o tipo *c_bool* da *ctypes*, é utilizada para representar a Pílula de Veneno utilizando um *booleano* com o tratamento adequado para a condição de corrida.

Durante a execução do método *start_scan* da classe *ExperimentController*, realizado na inicialização do sistema de aquisição, um objeto da classe *ACQController* é instanciado para criar os processos de aquisição e filtragem de dados. Estes processos são criados através da classe *Process* da biblioteca *Multiprocessing*. Durante a construção de um objeto dessa classe devem ser passados como argumentos o alvo (*target*), que é a função ou método que será executado e os respectivos argumentos (*args*). Em seguida, deve-se executar o método *start* que cria um novo processo que executa o método ou função alvo concorrentemente com o fluxo do processo principal.

Além da Pílula de Veneno, representada pelo tipo *Value* da biblioteca *Multiprocessing*, outra tipo de variável de memória compartilhada desta biblioteca é utilizada no sistema de aquisição, conforme será discutido: a *Queue*. A *Queue* é uma implementação do conceito de fila, ou seja, é uma estrutura de dados do tipo *First In, First Out* (FIFO), que significa “primeiro a entrar, primeiro a sair”. A figura 28 ilustra o funcionamento do sistema de aquisição.

O processo responsável pela aquisição dos dados é executado através do método *exec_acq* da classe *ACQController*. Entre os demais objetos relacionados às informações sobre a sequência de pulsos e a configuração do sistema, este método recebe também como argumento uma instância de *Queue* (*filter_queue*), que é utilizada para informar ao processo de filtragem a localização dos arquivos recém-adquiridos do espectrômetro através do

ACQServer. Para que a aquisição dos dados seja feita, uma instância da classe *SpecClient* é criada dentro do escopo do método executado por este processo.

Figura 28 - Sistema de aquisição do *ToRM Console*. Os dados são adquiridos do *ACQServer* pelo *SpecClient* através do protocolo FTP e enviados para o processo de filtragem. Concorrentemente, os dados filtrados são enviados ao sistema de visualização, que é executado no processo principal. Além disso, os dados são armazenados em memória no processo principal e gravados em disco no final da execução do experimento. Todas estas etapas são gerenciadas pela classe *ExperimentController*.

Fonte: Elaborada pelo autor.

Após estabelecer a comunicação, o método *execute_sequence* da *SpecClient* é executado. A primeira tarefa deste método é o envio dos comandos referentes ao início da execução do espectrômetro, levando em consideração qual módulo é executado. Também faz parte da funcionalidade deste método, detectar a localização do servidor através da interpretação do endereço do mesmo de modo que, se o servidor estiver rodando no mesmo computador que o *ToRM Console*, então apenas uma cópia de cada arquivo gerado pelo *ACQServer* no diretório '*ACQServerHome*' é feita para o diretório gerado pela inicialização do *scan* dentro subdiretório do '*CIERMag Workingplace*', '*data*'. Esta abordagem é realizada através do método *get_local_data*, da *SpecClient*. Caso contrário, a aquisição dos dados é feita através do protocolo FTP. Neste caso, o método *get_remote_data* é utilizado. Em ambos os casos, a abordagem é semelhante: observa-se o valor da Pílula de Veneno e verifica-se a disponibilidade do próximo arquivo^{xiii}.

^{xiii} Os arquivos gerados pelo *ACQServer* recebem nomes sequenciais (*data-1.h5*, *data-2.h5*, etc.).

O processo de filtragem é criado da mesma forma que o processo de aquisição, embora, além da fila de filtragem (*filter_queue*) ele também receba como argumento a fila de dados prontos (*ready_queue*). Os dados são lidos a partir da localização dos dados disponíveis na *filter_queue*. Estes dados são armazenados no formato *HDF5*, que será mais bem detalhado na seção 4.10. Como é utilizada uma representação complexa para a representação do sinal (sinal em fase e em quadratura), conforme discutido no capítulo 2, as partes reais e imaginárias são adquiridas no mesmo arquivo e cada par delas é convertido no tipo *complex*, nativo do *Python*. O arranjo completo desses dados é armazenado no *N-dimensional Numpy Array* (*numpy.ndarray*). Este tipo é bem consolidado entre as aplicações científicas utilizando *Python* e possui recursos que podem aumentar o desempenho das aplicações, tendo em vista algumas limitações do *Python* em relação a esse quesito. Em seguida, os dados são sujeitos a um processamento utilizando filtros passa-baixa do tipo *Finite Impulse Response* (FIR). O detalhamento desta etapa é feita na seção 4.9. Por fim, o processo de filtragem envia os dados filtrados para a fila *ready_queue*.

Do ponto de vista do fluxo do processo principal, o armazenamento em memória de todos os dados prontos obtidos a partir da *ready_queue* é feito com acompanhamento em tempo quase-real dos sinais filtrados, conforme mostrado na figura 29 e na figura 30. A visualização deve ser obrigatoriamente realizada no processo principal devido às restrições impostas pelo *PySide*. Entretanto, a plotagem contínua dos dados dá a impressão ao usuário de que a janela está paralisada, impossibilitando qualquer ação simultânea. Este problema é resolvido utilizando o método *processEvents()* da aplicação (*QtCore.QCoreApplication*), que processa os eventos pendentes da interface gráfica, intercalando entre os procedimentos de plotagem. Deste modo, o processo principal realiza as operações de visualização ao mesmo tempo em que possibilita o usuário interagir com a interface gráfica.

A biblioteca *Matplotlib* é utilizada através de um *backend*, que é uma adaptação para que se possa utilizá-la dentro do contexto do *PySide*. Deste modo, através da classe *FigureCanvasQTAgg* deste *backend*, que é herdeira da classe *QtWidget* do *PySide* é possível utilizar os recursos da *Matplotlib*. Recursos como *zoom* vertical e horizontal, deslocamento vertical e horizontal, por meio de teclas de atalhos ou interação utilizando o ponteiro do *mouse* e exportações de imagens foram desenvolvidos ou aproveitados dos recursos já existentes na biblioteca.

Conceitualmente, o produto final da execução do espectrômetro deve resultar na obtenção de uma matriz multidimensional, onde o número de dimensões e a quantidade de elementos são definidos pelo método. Entretanto, a ordem em que os elementos que formam

essa matriz são obtidos e o uso de médias deve ser levado em consideração. Mecanismos que serão discutidos na seção 4.8 são utilizados para o cálculo dos índices que correspondem às posições em que os dados serão inseridos na matriz. Através da classe *IndexWidget*, que constitui um elemento gráfico por ser subclasse da *QtWidget* do *PySide*, é possível acompanhar os índices dos últimos dados adquiridos com seus respectivos identificadores.

Figura 29 - Sistema de visualização do *ToRM Console* (*Viewport*) no domínio do tempo. É possível observar a parte real (azul) e imaginária (verde) e o módulo (vermelho) do sinal. No canto inferior direito está contido o *IndexWidget*, informando o índice do dado visualizado. No menu superior esquerdo estão, da esquerda para direita os botões de *zoom* vertical e horizontal, o botão de ajuste de escala no dado atual, o botão de auto escala, o botão de chaveamento entre domínio do tempo e frequência e o botão de chaveamento entre visualização do último dado ou da média entre eles. No menu inferior à direita estão os botões de deslocamento dos eixos com auxílio do ponteiro do *mouse*, a ferramenta de *zoom*, com a definição da região também através do *mouse*, o botão de configuração de ajuste do tamanho da área de visualização e, por último, o botão de exportação de figuras.

Fonte: Elaborada pelo autor.

Figura 30 - Sistema de visualização do *ToRM Console* (*Viewport*) no domínio da frequência. O cálculo da Transformada de Fourier é realizado através do pacote *numpy.fft*. O processamento é realizado apenas para visualização, não afetando os dados que serão armazenados em disco.

Fonte: Elaborada pelo autor.

4.8 Protocolo de aquisição, organização de dados e tipos especiais de *loops*

O protocolo de aquisição tem por finalidade armazenar todas as informações que alimentam o sistema de aquisição. Da mesma forma que todos os outros arquivos de configuração do sistema, ele também é um arquivo de texto contendo informações através da estrutura JSON. O índice mais externo é denominado '*modules*' e abaixo dele estão contidos os índices que recebem os nomes dos módulos existentes no método, além do índice '*executed_module*'. Este último informa o módulo que realmente foi executado e é preenchido pelo *ToRM Console*, uma vez que é através dele que o usuário realiza a escolha do módulo, conforme discutido na seção 4.5.

Três categorias de configurações estão contidas abaixo dos índices respectivos aos módulos: '*EntryPoint*', que contém o endereço de memória do módulo, de modo que o *ToRM Console* consiga enviar a requisição ao *ACQServer* para que ele seja utilizado; '*acquisition*', que possui informações relevantes ao sistema de filtragem e à requisição de dados ao *ACQServer*; e '*iterations*', que descreve a estrutura de *loops*, ou iterações, do respectivo módulo.

A categoria '*acquisition*' deve ser preenchida pelo *script* da validação desenvolvido pelo programador da sequência, conforme discutido na subseção 4.6, tendo em vista que todos

os valores de parâmetros necessários estão disponíveis para leitura nesta etapa. Os parâmetros relativos às configurações do *ACQServer* são necessários em função do mesmo não ter conhecimento detalhado sobre a estrutura do método de sequência de pulsos. As seguintes informações devem ser preenchidas nesta categoria:

- *samples*: número de amostras coletadas em cada iteração pelo *ACQServer*;
- *eff_samples*: número efetivos de amostras após o processo de filtragem (processo de decimação, discutido em 4.9);
- *samples_per_file*: número de amostras enviadas no mesmo arquivo. Bastante utilizado para visualização simultânea de múltiplos ecos;
- *repeat*: número total de iterações executadas pelo *ACQServer*;
- *clusters*: número de blocos de dados que serão adquiridos internamente pelo *ACQServer*. Quando o número de ecos obtidos é grande, pode haver sobrecarga de comunicação entre o espectrômetro e o *ACQServer*. Este parâmetro é utilizado para informar a quantidade de dados obtidos de uma só vez, de modo que a sobrecarga causada pelos protocolos de comunicação (*USB* ou *Gigabit Ethernet*) seja reduzida. Recomenda-se preencher este parâmetro com o mesmo número de que o de ecos;
- *acq_spectral_width*: taxa de amostragem dos dados do espectrômetro enviados pelo *ACQServer*. Como este valor pode ser alterado em função de modificações do *hardware*, ele deve ser fornecido;
- *spectral_width*: frequência de corte do filtro. Necessário para o sistema de filtragem e visualização^{xiv}.

^{xiv} No caso do sistema de visualização, ele é utilizado principalmente para gerar o eixo horizontal no domínio da frequência.

```

"iterations" : {
  "Average" : {
    "Echo" : {
      "Size" : 32.0,
      "Type" : "echoTrain",
      "Label" : "Echo"
    },
    "Size" : 128.0,
    "Type" : "average",
    "Label" : "Average"
  }
}

```

Figura 31 - Estrutura de *loops* ('iterations') contida no protocolo de aquisição do único módulo contido no método 'CPMG_E'. Os valores dos índices 'Size' de cada *loop* podem ser alterados por meio de parâmetros acessíveis à aba de parâmetros. Os demais valores são definidos no código F da sequência.

Fonte: Elaborada pelo autor.

Em relação à categoria '*iterations*', ela é gerada automaticamente pelo processo de compilação. Nela estão contidas as chaves que recebem os nomes dos módulos, que possuem a estrutura dos *loops* de cada um deles. A estrutura dos *loops* possuem as informações de hierarquia, o número de iterações em cada *loop*, o identificador (*Label*) e o tipo do mesmo, a ser discutido a seguir. Por exemplo, a sequência 'CPMG_E', utilizada neste trabalho (capítulo 6), apresenta a estrutura mostrada na figura 31 de *loops* em seu único módulo.

Por não conhecer a estrutura interna do método, o *ACQServer* envia todos os dados sequencialmente, separados em arquivos descritos pelos parâmetros contidos na categoria '*acquisition*' do protocolo de aquisição. Deste modo, o sistema de aquisição do *ToRM Console*, de algum modo, deve ser informado a respeito dos índices que identificam os dados que estão sendo recebidos para utilização de médias, visualização e armazenamento em memória e disco.

Outro fato importante é que o espectrômetro do CIERMag não limita o número de *loops* que podem ser utilizados, nem especifica os nomes que os identificam, conforme observado nos sistemas comerciais, onde normalmente esta estrutura é travada com número finito de *loops*. Em vez disso, são definidos os *tipos especiais de loops*. Os *loops* que possuem tipos especiais realizam operações automatizadas durante processo de aquisição. Desta forma, caso o programador queira utilizar os recursos oriundos dessa abordagem, ele deve apenas, no código da sequência, adicionar o parâmetro *Type* na declaração do *loop*.

Levando em consideração apenas a aquisição de um conjunto de *loops* ordinários, isto é, que não possuem tipos especiais, o problema da indexação já é existente. Por exemplo, se um dos módulos de uma sequência contém apenas dois *loops* ordinários denominados

‘*LoopExterno*’, com 3 iterações e ‘*LoopInterno*’, com 5 iterações, então o produto final será uma matriz 3x5, com o total de 15 elementos. Tuplas são utilizadas para identificar esses índices, uma vez que esse tipo de dado é nativo do *Python* (*tuple*). O valor inicial de cada entrada deve ser zero. A figura 32 ilustra a evolução dos índices para este caso.

Figura 32 - Evolução dos índices quando existem apenas os *loops* ‘*LoopExterno*’ com 3 iterações (vermelho) e ‘*LoopInterno*’ com 5 iterações (azul). O índice unidimensional (preto) varia de 0 a 14.

Fonte: Elaborada pelo autor.

Em cada iteração é realizada a leitura de uma determinada quantidade de amostras, conforme preenchido no índice ‘*samples*’ da categoria ‘*acquisition*’ do protocolo de aquisição. Esta dimensão não é considerada na geração dos índices e será omitida na maioria dos casos que serão apresentados a seguir. Para o caso contendo apenas *loops* ordinários, apenas a função *unravel_index* do *Numpy* é suficiente para solução do problema. Esta função recebe como parâmetro o valor do índice unidimensional, que representa a ordem do elemento e uma tupla com o formato da matriz. Por exemplo, no caso mostrado na figura 32, o formato seria ‘(3,5)’.

O primeiro tipo *loop* especial é o ‘*Average*’, que é referente à utilização de médias. Adaptando o exemplo anterior, será considerado agora o caso onde o ‘*LoopExterno*’ possui o tipo ‘*Average*’. Desta forma, a entrada referente ao ‘*LoopExterno*’ deve ser removida da tupla do índice multidimensional, de modo que o conjunto de amostras indexado com o mesmo valor para o ‘*LoopInterno*’ sejam somados, conforme mostrado na figura 33. Os dados são normalizados no final da execução de acordo com o número total de médias. A matriz final de dados ‘perde’ a dimensão referente à média, tendo o formato ‘(5)’ neste caso.

Figura 33 - Exemplo adaptado da figura 32 para o caso em que o 'LoopInterno' possui tipo 'Average'. O índice referente ao 'LoopInterno' é simplesmente removido.

Fonte: Elaborada pelo autor.

O segundo tipo especial de *loop* é o 'EchoTrain', que é utilizado em sequências onde deseja-se visualizar múltiplos ecos durante o tempo de aquisição, que é o caso, por exemplo, do método 'CPMG_E'. Se apenas *loops* ordinários forem utilizados, os ecos serão adquiridos separadamente e, portanto, apenas um eco será observado por vez. Desta forma, quando for utilizado um *loop* do tipo 'EchoTrain' deve-se fazer com que no *script* de validação o valor do 'samples_per_file' seja igual ao valor contido em 'samples' vezes o número total de ecos, de maneira que o ACQServer envie todos os ecos em um mesmo arquivo. Quando detectado um *loop* deste tipo, o ToRM Console filtra todos os ecos separadamente e os une novamente para fins de visualização. A dimensão relativa aos ecos é levada em consideração durante a criação da matriz final de dados.

O terceiro tipo especial de *loop* é o 'Split', utilizado em métodos de imagens que utilizam a técnica de fatiamento ou *slicing*. Cada fatia, ou *slice*, requer um determinado intervalo de tempo para ser adquirida. Quando o número de *slices* é relativamente grande, o tempo de repetição, que é o tempo entre o início de uma iteração e outra e possui inter-relações com outros parâmetros, pode ser menor do que o tempo total demandado pela aquisição de todos os *slices*. Essa situação deve causar o impedimento da execução do método durante a etapa de validação.

Desta forma, ao invés de adquirir todos os *slices* de uma só vez, podem-se obtê-los em dois ou mais experimentos distintos ou adicionar uma dimensão mais externa a todos os *loops*, de maneira a simular a execução de múltiplos experimentos em um mesmo *scan*. Este *loop* deve ser do tipo 'Split' e, por não possuir significado físico no experimento, deve ser

descartado na produção da matriz final. O *loop* correspondente aos *slices* deve possuir o tipo ‘*Slice*’, gerando mensagem de erro caso ele não exista simultaneamente. Este tipo de *loop* compreende normalmente a terceira dimensão espacial, quando utilizado num experimento *multislice*. A primeira dimensão é propriamente a de aquisição do sinal (recíproca da dimensão temporal da aquisição). A segunda neste caso de *multislice* é a codificação de fase, recíproca do pseudo-tempo que transcorre entre aplicações de gradientes de codificação de fase.

Para o caso do exemplo contendo apenas dois *loops* a solução é trivial: os índices são iguais aos índices unidimensionais. Sendo assim, três *loops* serão utilizados para demonstrar o algoritmo: o ‘*LoopExterno*’, que possui o tipo ‘*Split*’; o ‘*LoopIntermediário*’, que possui o tipo ‘*Slice*’; e o ‘*LoopInterno*’, do tipo ordinário. Por simplificação, todos possuem duas iterações, de forma que o número total de cortes desejados será, então, quatro, embora sejam adquiridos apenas dois por vez. A figura 34 mostra o perfil deste tipo de geração automática de índices.

Figura 34 - Geração de índices quando três *loops* são utilizados com tipos ‘*Split*’ (‘*LoopExterno*’, em vermelho), ‘*Slice*’ (‘*LoopIntermediário*’, em verde) e ordinário (‘*LoopInterno*’, em azul), em ordem decrescente de hierarquia. Todos os *loops* possuem duas iterações. As duas colunas representam a execução do primeiro (à esquerda) e segundo (à direita) ‘*Split*’, embora sua dimensão seja truncada. O número efetivo de *slices* será, portanto, quatro.

Fonte: Elaborada pelo autor.

Para obter este resultado, primeiramente é necessário calcular o índice multidimensional da mesma maneira mostrada nos exemplos anteriores. Em seguida, o índice referente aos *slices* é recalculado da seguinte forma:

$$s' = s + iN_{slices} \quad (49)$$

Onde s' é o novo índice referente ao *slice*, s é o índice original do *slice*, i é o índice do *split* e N_{slices} é o número total de *slices*. Por fim, o índice referente ao *loop* do tipo '*Split*' deve ser removido.

Os últimos tipos de *loops* especiais são referentes à utilização de codificação de fase, também utilizado em métodos de imagens. Para realizar a leitura da tabela de codificação de fase, que determina a ordem em que o *espaço-k* será varrido, os respectivos *loops* devem possuir os seguintes tipos: '*PhE1*', '*PhE2*' e '*PhE3*'. Cada um deles indica uma direção diferente e o uso simultâneo indica a dimensionalidade da imagem, que pode ser 2D ou 3D. Os exemplos de utilização são:

- 3D verdadeiro: primeira dimensão é a da aquisição, e as duas outras são codificações de fase, por exemplo, '*PhE1*' e '*PhE2*';
- *Multislice*: primeira dimensão é a da aquisição, e as duas outras são codificações de fase '*PhE1*' e '*Slice*';
- O terceiro caso é raramente utilizável, mas refere-se a uma técnica de aquisição em que alguns pontos do sinal são adquiridos e acumulados, mediante codificação espacial de fase nas três dimensões: '*PhE1*', '*PhE2*' e '*PhE3*'.

Além disso, alguns métodos, também relacionados à técnica de obtenção de imagens, utilizam múltiplos ecos onde cada um deles possui codificação diferente de fase. Deste modo, é desejável observar todos os ecos simultaneamente e, durante o armazenamento dos dados, remover a dimensão referente aos ecos, de maneira que cada um deles defina apenas a respectiva codificação de fase. Neste caso, deve-se utilizar um dos seguintes tipos: '*EchoTrainPhE1*', '*EchoTrainPhE2*' ou '*EchoTrainPhE3*'. Eles devem ser utilizados simultaneamente com os respectivos *loops* com tipos de codificação de fase, mencionados anteriormente. O algoritmo é análogo ao utilizado no tipo '*Split*', embora os *loops* de codificação de fase devam possuir hierarquia mais alta em relação aos ecos. A dimensão artificial introduzida pelos múltiplos ecos é posteriormente colapsada na dimensão de codificação de fase correspondente.

Todos os tipos de *loops* vistos nessa subseção foram desenvolvidos através classe *Data_Organization*. A estratégia utilizada é, após a leitura do protocolo de aquisição, todas as informações relevantes como as posições dos tipos de *loops* especiais e o número de iteração em cada *loop* são mantidas e, além disso, métodos distintos para a obtenção de índices para visualização, armazenamento em memória e armazenamento em disco são utilizados, uma vez que eles não possuem sempre o mesmo comportamento. Como a visualização em tempo de

aquisição necessita receber normalização a cada novo conjunto de dados se o recurso de realização de médias for utilizado, também é possível obter apenas o número de médias corrente, de modo a normalizar apenas o último dado visualizado^{xv}.

Os diferentes tipos especiais de *loops* podem ser utilizados simultaneamente, contanto que obedeçam as restrições mencionadas anteriormente.

4.9 Filtros

Os filtros são importantes para o sistema de aquisição por serem responsáveis por contribuir com a remoção de informação fora da região espectral de interesse e, conseqüentemente, melhorar a relação sinal-ruído do sinal. Além disso, a teorema da amostragem de *Nyquist* e *Shannon* prevê a seguinte relação entre a componente com a maior frequência contida do sinal, f_{max} , e a taxa de amostragem $f_s = 1/T$, onde T é o tempo amostragem^{xvi}: (34)

$$f_{max} < \frac{f_s}{2} \quad (50)$$

Ou seja, a taxa de amostragem deve ser duas vezes maior do que a maior frequência presente no sinal para que não haja perda de informação. Deste modo, como a banda espectral de um experimento de ressonância magnética nuclear é normalmente prevista, é possível utilizar a quantidade mínima de pontos que devem ser armazenados, através da escolha adequada da frequência de amostragem, reduzindo o espaço em disco para armazenamento e aumentando a velocidade nos procedimentos de visualização e processamento.

Em princípio, a alteração da taxa de amostragem poderia ser controlada por *hardware*^{xvii}. Porém, a frequência de amostragem com que os dados enviados pelo espectrômetro do CIERMag, embora possa ser alterada futuramente, é fixada atualmente em 1 *Mega Samples Per Second* (MSPS)^{xviii}, no intervalo de frequências entre -500kHz e +500kHz^{xix}. Ainda que existam filtros implementados por *hardware*, é muito mais fácil obter

^{xv} A matriz de dados contidas em memória (*data_buffer*) só pode ser normalizada no final da execução.

^{xvi} O tempo de amostragem é igual ao intervalo temporal entre duas amostras consecutivas do mesmo sinal.

^{xvii} Obedecendo, obviamente, as limitações tecnológicas dos digitalizadores.

^{xviii} 1 MSPS é equivalente a 10^6 amostragens por segundo.

^{xix} A taxa de amostragem do digitalizador do espectrômetro é de 50 MSPS. Entretanto, pré-processamentos de filtragem e decimação do sinal são aplicados previamente no *hardware* de maneira que a taxa de amostragem com que o dados são enviados ao *ACQServer* é de 1 MSPS.

flexibilidade na implementação dos mesmos via *software*. Além disso, a combinação entre as duas abordagens não fica impedida, tendo em vista a possibilidade de aplicá-los em cascata.

Os filtros utilizados no sistema de aquisição do *ToRM Console* são filtros passa-baixa do tipo FIR, conforme mencionado anteriormente. Ideal e analiticamente, um filtro passa-baixa poderia ser interpretado pela equação abaixo^{xx}:

$$s_{filtrado}(t) = \mathcal{F}^{-1}\{\mathcal{F}\{s(t)\}B(\omega)\} = \mathcal{F}^{-1}\{S(\omega)B(\omega)\} \quad (51)$$

Onde $s(t)$ é o sinal original, a transformada de Fourier é dada pelo operador $\mathcal{F}$ e $B(\omega)$ é uma função com valor unitário na região de frequência de interesse, delimitada pela frequência de corte, f_c , tal que:

$$B(\omega) = \begin{cases} 1, & |\omega| \leq 2\pi f_c \\ 0, & |\omega| > 2\pi f_c \end{cases} \quad (52)$$

Ou seja, se a transformada de Fourier do sinal, $S(\omega)$, for multiplicada por uma função que anule toda a parte fora da região de interesse, deixando intacta a região interior, a transformada inversa de Fourier pode ser utilizada para obter o sinal filtrado sem a presença das componentes indesejadas. Além disso, é possível mostrar que a equação 51 é, na verdade, uma operação de *convolução* entre as funções $s(t)$ e $b(t) = \mathcal{F}^{-1}\{B(\omega)\}$:

$$s_{filtrado}(t) = \mathcal{F}^{-1}\{S(\omega)B(\omega)\} = (s * b)(t) \quad (53)$$

A versão discreta da equação acima possui a seguinte forma:

$$s_{filtrado}[k] = \sum_{i=0}^N b[i]s[k-i] \quad (54)$$

Onde N é a ordem do filtro, que indica a quantidade, $N + 1$, de coeficientes do filtro $b[i]$, também denominados *taps* e $s_{filtrado}[k]$ e $s[k]$ são elementos dos vetores que representam os sinais discretos filtrado e original, respectivamente. Neste caso, todos os elementos que constituem a filtragem estão no domínio do tempo.

A transformada inversa da função $B(\omega)$ para o caso mostrado na equação 52 resulta na função abaixo:

^{xx} Os fatores de normalização, que dependem da definição da transformada de Fourier, estão sendo omitidos.

$$b(t) = \mathcal{F}^{-1}\{B(\omega)\} = \frac{2f_c \sin(2\pi f_c t)}{2\pi f_c t} = 2f_c \text{sinc}(2f_c t) \quad (55)$$

E, para o caso arbitrário $f_c = 1$, por exemplo, possui a seguinte forma no intervalo de tempo $[-3, 3]$:

Figura 35 - Formato da função $b(t)=2f_c \text{sinc}(2f_c t)$ no intervalo $[-3, 3]$ para o caso $f_c = 1$ utilizando 1000 pontos.

Fonte: Elaborada pelo autor.

Esta função é definida em todo o eixo temporal e, por este motivo, ao tentar representá-la numericamente o seu truncamento ocorre significativamente em alguns casos, como pode ser visto na figura 35. A parte não representada por este conjunto de pontos pode causar artefatos significativos no sinal filtrado. Desta forma, o projeto de um filtro é baseado em sua ordem e escolha adequada dos coeficientes. Estes coeficientes podem ser encontrados na literatura. (35)

No sistema de filtragem desenvolvido no *ToRM Console* todas as operações são implementadas com o uso da biblioteca *Numpy* e *Scipy*. Para a obtenção dos coeficientes adequados, é utilizado da função *firwin* do pacote *scipy.signal*. Através do identificador do filtro^{xxi}, número de coeficientes, frequência de corte e frequência de Nyquist^{xxii} é retornado um vetor, do tipo *numpy.ndarray*, que representa os coeficientes utilizados. A operação de convolução é feita através da função *convolve* do *Numpy*.

^{xxi} Estes coeficientes também são denominados ‘funções janelas’.

^{xxii} A frequência de Nyquist é definida pelo valor da taxa de amostragem quando existe uma igualdade na equação 50. Essa frequência define a menor taxa de amostragem para que não haja perda de informação.

Embora todos os recursos necessários estejam aparentemente disponíveis, algumas particularidades foram incluídas na implementação. A primeira delas leva em consideração a existência de múltiplos ecos^{xxiii} no mesmo conjunto de dados. Esses ecos devem ser filtrados separadamente e, portanto, o conjunto de dados é dividido de acordo com o tamanho do grupo (*group_size*) através da função *split* do *Numpy*. Em seguida, outra dificuldade foi encontrada em função dos efeitos de borda da filtragem, conforme discutido na legenda da figura 36 e também mostrado no caso apresentado na figura 37.

O resultado da convolução entre um conjunto de dados de N pontos representando o sinal e os M coeficientes do *kernel* do filtro possui $N + M - 1$ pontos. É sabido que apenas $N - M + 1$ pontos são válidos através da implementação do *numpy.convolve*. Esta implementação fornece três modos operação: *'full'*, que retorna todos os $N + M - 1$ pontos, *'valid'*, que retorna apenas $N - M + 1$ pontos válidos e *'same'* que retorna N pontos, embora ainda apresente $M - 1$ pontos com efeitos de borda. Nenhum desses modos retorna os desejados N pontos válidos.

Sendo assim, a solução proposta por este trabalho consiste na concatenação de $(M - 1)/2$ pontos em cada extremidade com valores iguais aos das respectivas bordas no sinal. A operação de convolução deve ser, portanto, utilizada no modo *'same'*. Após a filtragem, os $M - 1$ pontos adicionais são removidos das extremidades, restando N pontos válidos. A figura 38 ilustra o processo com um sinal cossenoidal, enquanto a figura 39 com um decaimento exponencial. Em ambos os casos foram utilizados $N = 1000$ e $M = 255$ com a janela *'Hanning'*.

^{xxiii} Normalmente, o agrupamento de dados é utilizado para representação e visualização simultânea de múltiplos ecos. Entretanto, o sistema de filtragem analisa apenas o tamanho do grupo (*group_size*), independentemente do tipo de agrupamento.

Figura 36 - Problema observado após a aplicação de filtro FIR por meio de convolução com janela ‘Hanning’. O sinal (azul) representa a função $\sin(\omega t)$, com $\omega = 1$ no intervalo $[0, 2\pi]$. A segunda curva representa a soma do sinal com um ruído gaussiano (vermelho). Embora o ruído de alta frequência tenha sido removido com a aplicação de um filtro FIR com frequência de corte $\omega_c = 1.1\omega$, o sinal filtrado (verde) apresenta problemas relacionados aos efeitos de borda da filtragem. O número de coeficientes do filtro foi de $M = 255$ e o modo “same” do módulo `numpy.convolve` foi utilizado para remoção de $M - 1$ pontos inválidos pelo processo de convolução.

Fonte: Elaborada pelo autor.

Figura 37 - O mesmo problema observado na figura 36, porém utilizando a função $\exp(-t)$ no intervalo $[0, 5]$. O ruído de alta frequência é novamente removido, porém os efeitos transientes do filtro são encontrados na extremidade esquerda da curva do sinal filtrado. O mesmo número de coeficientes foi utilizado (255) e a filtragem também foi utilizada no modo “same”.

Fonte: Elaborada pelo autor.

Figura 38 - Solução para o problema relacionado aos efeitos de bordas da filtragem. O sinal original (azul) representa a função $\sin(\omega t)$, com $\omega = 1$ no intervalo $[0, 2\pi]$ com a adição de ruído gaussiano. Em seguida, são concatenados ao sinal pontos com os mesmos valores das bordas (vermelho). O processo de filtragem é realizado aparentando efeitos de bordas fora da região realmente pertencente ao sinal (verde). Por fim, a remoção dos pontos excedentes é realizada (marrom).

Fonte: Elaborada pelo autor.

Figura 39 - Solução aplicada a um decaimento exponencial ruidoso. O procedimento é análogo ao apresentado na figura 38.

Fonte: Elaborada pelo autor.

Após a filtragem, o conjunto de dados possuirá mais pontos do que ele realmente necessita para representar o sinal sem perda de informação, conforme discutido acerca do

teorema da amostragem. Sendo assim, o conjunto de dados passa por um processo denominado decimação, que reduz a quantidade de pontos por um fator inteiro D , tal que:

$$D = \frac{f_s}{f_c} \rightarrow N' = \frac{N}{D} \quad (56)$$

Onde N' é o número de pontos após a decimação, f_c é a frequência de corte do filtro e f_s é a taxa de amostragem dos dados enviados pelo espectrômetro ($f_c < f_s$).

Quando o número de ecos contido no mesmo conjunto de dados é muito grande, alguns problemas em relação ao desempenho foram percebidos. Sendo assim, novamente com o auxílio da biblioteca *Multiprocessing* foram aplicadas técnicas de processamento paralelo para reduzir o tempo de filtragem. Como os diversos ecos podem ser filtrados independentemente são criadas diversas *threads* através da classe *ThreadPool* do pacote *multiprocessing.pool*. O método *filter_* da classe *Filter* do sistema de processamento é enviado com parâmetro do método *map* da classe *ThreadPool*, de maneira que os diversos ecos possam ser filtrados paralelamente, utilizando o poder de processamento de todos os núcleos do processador.

4.10 Armazenamento de dados

Após o término da execução do sistema de aquisição do *ToRM Console*, os dados são gravados em disco no formato *Hierarchical Data Format* (HDF), na sua versão *HDF5*. Este formato foi projetado para armazenar grandes quantidades de dados numéricos. Além da permissão de uso deste formato, os responsáveis pelo desenvolvimento do *HDF5* fornecem gratuitamente bibliotecas para leitura e escrita nas linguagens *Java*, *MATLAB/Scilab*, *Octave*, *IDL*, *Python* e *R*. A organização dos arquivos é baseada em grupos (*Groups*) e conjunto de dados (*Datasets*).

Todos os dados são armazenados no mesmo arquivo, nomeado '*final_data.h5*', no diretório criado durante a execução do *scan* no interior do '*CIERMag Workingplace*'. Um grupo nomeado '*AcquiredData*' contém os diversos *Datasets* que são identificados pelos respectivos índices. Cada conjunto de amostra possui duas colunas contendo as partes reais e imaginárias da representação complexa do sinal.

Além disso, é possível visualizar os dados através do *software* gratuito HDFView, conforme ilustrado na figura 40. (36)

Figura 40 - Visualização de conjunto de dados obtidos através do *ToRM Console* pelo *software* HDFView. É possível observar os valores numéricos, bem como obter visualização gráfica.

Fonte: Elaborada pelo autor.

4.11 Documentação

O tipo de documentação escolhido para essa aplicação apoia-se no gerador automático de documentações *Sphinx*. (37) Através de *docstrings* do *Python*, que são textos (*strings*) inseridos logo após definição de classes, métodos e funções, esta ferramenta permite a geração de documentação em diversos formatos. O formato utilizado atualmente é o *HyperText Markup Language* (HTML), que permite com que a documentação possa ser acessada por qualquer navegador *web*, facilitando as consultas.

O sistema de referências cruzadas permite ligações entre as funções, métodos e classes desenvolvidos, além de permitir a inserção de textos adicionais contendo informações acerca do trecho do código ou funcionamento do sistema. Além disso, é possível realizar buscas e acessar as partes da documentação por meio de índices.

5 TORM PROCESSING API

5.1 Introdução

Até o momento foram detalhadas as etapas que antecedem a execução de experimento, bem como os outros procedimentos que ocorrem durante a aquisição dos dados, onde os protocolos relacionados com estas etapas são, respectivamente, o protocolo de execução e o protocolo de aquisição.

Durante a execução do sistema de aquisição do *ToRM Console* os dados passam por uma etapa intermediária, denominada pré-processamento, na qual ocorrem a filtragem e organização dos dados. Entretanto, depois que os dados foram adquiridos, é necessária a realização de outra etapa: o pós-processamento. O pós-processamento está relacionado fortemente com o tipo de aplicação do experimento executado. Sendo assim, é no final da etapa de pós-processamento que as informações realmente desejadas em um experimento de RM serão obtidas, seja ela uma imagem ou um conjunto de pontos, que pode ser um espectro de frequências ou alguma outra curva em que se pode, por exemplo, obter valores de tempos de relaxação longitudinal ou transversal. A figura 41 descreve as diversas etapas que os sistemas desenvolvidos neste trabalho realizam do ponto de vista das operações sobre os dados e a obtenção de informação.

Para efetuar o pós-processamento pode-se utilizar diversas ferramentas disponíveis por terceiros, embora, como mencionado anteriormente, esta abordagem pode se tornar limitada e ainda requerer adaptações ou conversões dos dados para que os mesmos possam ser utilizados por estas aplicações. Algumas tarefas como, por exemplo, a leitura de arquivos *HDF5*, utilizados para armazenamento dos dados, necessita do uso de bibliotecas e a compreensão detalhada deste formato. Além disso, o acesso aos protocolos de execução e aquisição, ferramentas para visualização e operações numéricas serão certamente necessários, tendo em vista o conjunto de aplicações de RM.

Deste modo, levando em consideração um dos objetivos principais deste trabalho, que é o de criar um espectrômetro digital de RM flexível, a abordagem escolhida foi a utilização de *scripts* em *Python*, onde o desenvolvedor do método de sequência de pulsos ou alguém que compreenda bem o funcionamento do mesmo possa programar todas as operações que devem ocorrer durante a etapa de pós-processamento. Este capítulo aborda as principais

características de uma *Application Programming Interface* (API), ou seja, uma Interface de Programação de Aplicativos, que é uma proposta para fornecer padrões e funcionalidades para utilização dos recursos mencionados anteriormente. Por fim, exemplos de uso são mostrados para o melhor entendimento desta ferramenta.

Figura 41 - Diversos estágios nos quais os dados adquiridos passam até a obtenção das informações resultantes. Inicialmente os dados são adquiridos pelo *ACQServer* e enviados ao *ToRM Console* (1), onde são filtrados (2), organizados (3) e armazenados em disco junto aos protocolos de aquisição e execução (4). Por fim, a execução do *script* de pós-processamento realiza as operações necessárias para obtenção da informação resultante, podendo ser uma imagem, uma curva, um número ou qualquer outro tipo de saída em função da aplicação (5).

Fonte: Elaborada pelo autor.

A *ToRM Processing API*, que foi implementada ao longo do desenvolvimento deste trabalho, é uma biblioteca que fornece um conjunto de operações necessárias para leitura e escrita de dados e protocolos, visualização de gráficos e imagens. O principal objetivo é fazer com que o programador possa realizar tarefas de automatização de pós-processamento sem a necessidade de conhecer todos os detalhes técnicos acerca do conjunto de informações gerado pelo sistema de aquisição. Embora não seja obrigatório, é recomendado o uso explícito da biblioteca *Numpy*, tendo em vista que a *ToRM Processing API* utiliza o formato *Numpy Array* para representação dos dados, além da vasta quantidade de operações numéricas otimizadas que são providas por esta biblioteca. Além disso, é recomendado ao programador utilizar, em alguns casos, as ferramentas de processamento de sinais da biblioteca *Scipy*.

Embora tenha sido utilizada a abordagem de orientação a objetos para desenvolver

esta API, deve-se levar em consideração a capacidade de suportar diversos paradigmas de programação da linguagem *Python*. Sendo assim, o programador dos *scripts* de pós-processamento poderá utilizar as técnicas de programação que lhes são mais adequadas ou convenientes, de maneira a evitar maiores restrições em relação ao conhecimento de programação do mesmo, para criação destes *scripts*. Além disso, métodos mais avançados, que não serão discutidos aqui, admitem a utilização de trechos de códigos escritos em linguagem C ou C++ dentro de um *script Python*, permitindo o reaproveitamento de códigos escritos nestas linguagens e, em alguns casos, aumento de desempenho.

5.2 Dependências, distribuição e instalação.

A *ToRM Processing API* é uma biblioteca que pode ser utilizada em qualquer computador que ofereça suporte à linguagem *Python* e às demais bibliotecas utilizadas por esta API, que nesta seção serão denominadas por dependências. Deste modo, a restrição em relação à portabilidade com os diversos sistemas operacionais existentes está relacionada com a disponibilidade do conjunto destas dependências. Atualmente as seguintes bibliotecas fazem parte deste conjunto: *PySide*, *PyQtGraph*, *h5py*, *Numpy*, *Scipy* e *python-opengl*. (38)

Não é necessário instalar ou executar o *ToRM Console*, bem como as demais aplicações do espectrômetro CIERMag para que o programador possa utilizar esta API. Sendo assim, é possível realizar processamento *off-line* dos dados em qualquer outro computador, embora o *ToRM Console*, através de integração com a *ToRM Processing API*, permita automatização da execução dos *scripts* de pós-processamento.

A biblioteca padrão do *Python*, *Distutils*, permite que versões para distribuição possam ser criadas. Para isso foi necessária a criação de um *script*, denominado por *setup.py*, que deve fazer parte do diretório raiz da biblioteca. Este script é responsável por configurar o conjunto de arquivos que serão responsáveis por compor a versão de distribuição e, quando executado, atua de acordo com parâmetros passados por linha de comando. Quando o parâmetro '*sdist*' é utilizado, é criada uma versão que contém uma cópia de todos os arquivos informados no *setup.py* e uma cópia dele mesmo. Quando este mesmo *script* é executado com o parâmetro '*install*', a biblioteca é adicionada ao diretório de bibliotecas instaladas no *Python*, tornando-se, a partir deste momento, disponível para utilização a partir do comando *import* em qualquer *script*.

Esta abordagem é normalmente encontrada em diversas aplicações *Python* e é a mais comum atualmente. No entanto, a instalação de programas por parte de usuários do sistema operacional *Microsoft Windows* normalmente é realizada através de ‘instaladores’, que são arquivos executáveis que tem como objetivo utilizar interfaces gráficas para auxiliar o usuário. A biblioteca *Distutils*, nativa do *Python*, oferece recursos para criar automaticamente estes instaladores através do *setup.py* com o parâmetro ‘*bdist_wininst*’. Deste modo, é possível instalar o *ToRM Processing API* através deste executável gerado nos sistemas *Microsoft Windows*, além da maneira abordada anteriormente. Outros tipos de formatos para os demais sistemas operacionais, como é o caso dos pacotes RPM, pacotes Debian (*deb*), entre outros, também podem ser criados através do *setup.py*, embora nenhum deles tenha sido utilizado neste trabalho. (39)

5.3 Estrutura dos arquivos de saída de um experimento

Para que os dados adquiridos em um experimento de ressonância magnética possam ser totalmente compreendidos ou reconstruídos, algumas informações além, obviamente, do conjunto de dados, representados por números complexos, obtidas pelo *ToRM Console* devem ser armazenadas. Estas informações são referentes ao conjunto de parâmetros do método e parâmetros do sistema contendo seus respectivos valores no momento da execução, bem como à estrutura de iterações utilizada no desenvolvimento da sequência de pulsos. Todas estas informações estão contidas no protocolo de execução, que contém todos os parâmetros de sistema e do método, e no protocolo de aquisição, que contém a estrutura das iterações da sequência de pulsos, bem como outras informações utilizadas pelo sistema de aquisição de dados. Após a execução de um experimento, o *ToRM Console* cria um diretório, identificado com o nome do método e o horário em que o mesmo foi executado dentro do diretório ‘data’, contido na pasta do usuário (‘*CIERMag Workingplace*’) autenticado. Este diretório, simbolicamente nomeado a seguir por ‘*OUTPUT_PATH*’, possui a estrutura indicada pela figura 42:

Figura 42 - Estrutura do diretório onde se encontram os arquivos produzidos pelo sistema de aquisição do ToRM Console.

Fonte: Elaborada pelo autor.

Todos os subdiretórios e arquivos contidos dentro de '*OUTPUT_PATH*' são descritos a seguir:

- *prc_script.py*: este arquivo é uma cópia do *script* de pós-processamento padrão que está contido no diretório do método executado. Sua finalidade é realizar, através da linguagem *Python*, todas as tarefas de automatização de pós-processamento, incluindo visualização, por meio de auxílio da *ToRM Processing API*, que permitirá acesso aos arquivos que serão discutidos subsequentemente;
- *raw_data*: neste subdiretório estão contidos todos os arquivos de formato *HDF5* contendo os dados adquiridos pelo espectrômetro através do *ACQServer*. Estes dados não foram processados pelo filtro FIR do sistema de aquisição do *ToRM Console* e, portanto, representam os dados recebidos diretamente pelo *ACQServer*;
- *final_data.h5*: este arquivo, em formato *HDF5*, contém todos os dados obtidos no experimento. Estes dados foram processados pelo filtro FIR do sistema de aquisição do *ToRM Console* e já estão reorganizados para fins da criação da matriz utilizada no *script* de pós-processamento;
- *acq_protocol.json*: este arquivo é uma cópia do protocolo de aquisição que está contido no diretório do método executado. Este protocolo de aquisição, gerado na etapa de preparação do *scan*, contém todas as informações referentes à estrutura das iterações contidas na sequência de pulsos, além de outras informações utilizadas pelo sistema de aquisição para organização dos dados, conforme discutido na seção 4.8;

- *exec_protocol.json*: este arquivo é uma cópia do protocolo de execução, contido no diretório do método executado. Todos os valores dos parâmetros de sistema e dos parâmetros do método no momento da realização do *scan* estão contidos neste arquivo, conforme discutido na seção 4.5.

5.4 Managers

Todos os recursos da *ToRM Processing API* são acessados através de *Managers*. *Managers* são classes que fornecem um conjunto específico de operações através de seus respectivos métodos. Atualmente estão implementados os seguintes *Managers*: *DataManager*, *ProtocolManager*, *PathManager* e *WidgetManager*. O objetivo desta seção é detalhar cada um deles a apresentar exemplos de uso.

Os principais métodos e propriedades destas classes também serão apresentados aqui, embora esta não seja a melhor maneira de documentá-los em longo prazo, tendo em vista possíveis alterações desta API no futuro. Foram utilizadas as mesmas ferramentas que o *ToRM Console* (seção 4.11) utiliza para criação da documentação desta biblioteca. Sendo assim, as discussões em relação às especificações destes métodos terão validade garantida até a data de apresentação deste trabalho, sendo necessária a consulta da documentação oficial futuramente.

5.4.1 ProtocolManager

Esta classe é responsável por fornecer acesso aos protocolos de execução e aquisição. Através dela é possível acessar parâmetros de um determinado protocolo de execução de um método de sequência de pulsos e obter informações sobre a estrutura, organização e outras características pertinentes aos dados obtidos em um experimento através dos elementos contidos no protocolo de aquisição. O programador deve indicar os caminhos para os protocolos de aquisição e execução, que pode ser obtido com auxílio da classe *PathManager*, que será detalhada na subseção 5.4.3.

Embora a leitura dos protocolos, que estão escritos em formato JSON, possa ser feita sem o auxílio desta classe por meio da biblioteca *json*, padrão do *Python*, é recomendado a utilização de instâncias da classe *ProtocolManager* no desenvolvimento de *scripts* de pós-processamento, tendo em vista que toda modificação futura na estrutura do protocolos será refletida nesta classe. Sendo assim, modificações nos *scripts* de pós-processamento serão eliminadas ou, pelo menos, reduzidas em função do impacto de alterações nos protocolos.

Os métodos e propriedades^{xxiv} implementados até o presente momento desta classe serão apresentados brevemente a seguir.

- *exec_protocol* (propriedade): retorna um dicionário (*dict*), que é um tipo nativo do *Python*, contendo todos os parâmetros e respectivos valores dentro dos grupos ‘*methodology*’, ‘*system*’, ‘*real_time*’ e ‘*globais*’ utilizados no protocolo de execução carregado previamente. Gera uma exceção caso não haja protocolo de execução carregado previamente;
- *acq_protocol* (propriedade): semelhante ao *exec_protocol*, retorna um dicionário contendo as informações do protocolo de aquisição. Também gera uma exceção quando não há protocolo de aquisição carregado previamente;
- *open_exec_protocol*: carrega um protocolo de execução. Deve receber como parâmetro o caminho para o mesmo. Gera exceção caso o arquivo JSON esteja formato erroneamente ou o caminho esteja inválido;
- *open_acq_protocol*: análogo ao *open_exec_protocol* para o caso do protocolo de aquisição;
- *open_protocols*: permite o carregamento simultâneo dos protocolos de aquisição e execução. Os parâmetros devem ser informados através dos argumentos indexados *acq_path* e *exec_path*, que são objetos do tipo *string* contendo o caminho dos respectivos protocolos. Este método é utilizado internamente pelos métodos *open_exec_protocol* e *open_acq_protocol*, embora o programador também possa utilizá-lo;
- *get_parameter*: retorna o valor de um determinado parâmetro. Recebe como argumento o nome do parâmetro e a categoria, que pode ser omitida. Se a categoria for omitida, assume-se que o parâmetro a ser obtido é da categoria ‘*methodology*’.

Este método gera uma exceção se o protocolo de execução não estiver carregado;

^{xxiv} Uma propriedade (*property*) no *Python* substitui os métodos *get* e *set* utilizados em outras linguagens de programação orientadas ao objeto (*C++*, *Java*, *etc.*). A forma de leitura e escrita é semelhante ao de um atributo qualquer, embora esta abordagem permita o uso de encapsulamento, possibilitando o tratamento dos valores de entrada e saída da propriedade.

- *set_parameter*: altera o valor de um determinado parâmetro no protocolo de execução carregado. Deve-se receber como parâmetro nome do parâmetro e a categoria. Se a categoria não for informada, assume-se a categoria ‘*methodology*’. Se nenhum protocolo de execução estiver carregado, é gerada uma exceção;
- *list_parameters*: retorna uma lista contendo todos os parâmetros de uma determinada categoria. A categoria deve ser informada como argumento deste método, caso contrário, será assumida a categoria ‘*methodology*’. Este método também gera uma exceção se o protocolo de execução não estiver carregado;
- *save_acq_protocol*: salva protocolo de aquisição. É gerada uma exceção se não houver protocolo de aquisição carregado;
- *save_exec_protocol*: análogo ao *save_acq_protocol* para o caso do protocolo de execução;
- *help*: imprime na tela instruções para o uso desta classe.

5.4.2 DataManager

Esta classe é responsável por auxiliar a leitura de dados obtidos em um experimento. Utilizando-se recursos da biblioteca *h5py* e informações contidas no protocolo de aquisição, esta classe carrega os dados experimentais contidos no arquivo de formato *HDF5* na memória por meio de um objeto do tipo *Numpy Array* (*numpy.ndarray*). Os métodos implementados até o presente momento serão discutidos a seguir:

- *read_data*: retorna um objeto do tipo *numpy.ndarray* contendo uma matriz multidimensional de pontos complexos representando os dados obtidos em um determinado experimento e registra estes dados através do método *register_data*, que será discutido a seguir. Este método recebe como parâmetro o caminho para o arquivo *final_data.h5* e o caminho para o protocolo de aquisição. É possível, ao invés de utilizar o caminho para o protocolo de aquisição, enviar o objeto retornado pela propriedade *acq_protocol* da classe *ProtocolManager*. Este método pode gerar diversas exceções relacionadas aos problemas com a leitura e interpretação do arquivo *final_data.h5* e do protocolo de aquisição;
- *register_data*: registra um objeto do tipo *numpy.ndarray* dentro de uma lista,

declarada estaticamente, ou seja, que pode ser acessada através da classe *DataManager*, ao invés de uma instância da mesma. É aguardado como argumento o objeto do tipo *numpy.ndarray* e uma descrição para o mesmo, representada através do tipo *string*. O uso desta funcionalidade será discutido adiante;

- *save*: salva os dados de um determinado objeto do tipo *numpy.ndarray* contendo valores complexos em um arquivo *HDF5*. Para fins da criação e nomenclatura dos grupos dentro dos arquivos *HDF5*, serão observadas as dimensões do objeto que contém os dados. Este método recebe como parâmetro o objeto do tipo *numpy.ndarray* a ser salvo e o destino do novo arquivo *HDF5* que, se for omitido, será armazenado no diretório corrente do *script* que está sendo executado;
- *help*: imprime na tela instruções para o uso desta classe.

5.4.3 PathManager

Esta classe é responsável por facilitar a obtenção dos caminhos dos subdiretórios e arquivos contidos em ‘*OUTPUT_PATH*’ através das propriedades que serão discutidas abaixo. Durante a criação da instância desta classe, deve-se passar como argumento do construtor desta classe apenas o diretório raiz dos arquivos resultantes do sistema de aquisição do *ToRM Console*, ou seja, apenas o caminho para ‘*OUTPUT_PATH*’. Caso este argumento seja omitido, será considerado o diretório corrente de onde o *script* está sendo executado.

Os métodos implementados atualmente são:

- *acq_protocol_path* (propriedade): retorna o caminho para o protocolo de aquisição, levando em consideração a estrutura do ‘*OUTPUT_PATH*’;
- *exec_protocol_path* (propriedade): semelhante à propriedade anterior para o caso do protocolo de execução;
- *data_path* (propriedade): análogo aos anteriores para o caso do caminho de *final_data.h5*;
- *help*: imprime na tela instruções para o uso desta classe.

É importante ressaltar a importância desta classe quando alterações futuras na estrutura do ‘*OUTPUT_PATH*’ que, ainda que sejam evitadas ao máximo, podem acontecer ao longo do tempo de vida deste projeto. Sendo assim, caso estas alterações ocorram, elas

deverão ser refletidas na implementação da classe *PathManager*, evitando ou, pelo menos, reduzindo modificações no código desenvolvido utilizando esta API.

5.4.4 *WidgetManager*

Esta classe é responsável por prover um conjunto de ferramentas para facilitar o desenvolvimento da interface gráfica durante a etapa de pós-processamento. O principal objetivo deste *Manager* é agrupar instâncias de elementos gráficos do *PySide* através de uma janela interativa, que será discutida na próxima seção e que é implementada através da classe *InteractiveWidget*. Os métodos abaixo viabilizam esta tarefa:

- *begin_qtapp*: Retorna uma aplicação *PySide*. Este método é apenas uma maneira alternativa de criar um instância da classe *PySide.QtGui.QApplication*. Em função da arquitetura da biblioteca *PySide*, é necessária a criação de uma instância dessa classe antes da utilização de qualquer outra classe que implemente um elemento gráfico. O principal objetivo deste método é fornecer uma maneira alternativa de criar esta instância sem a necessidade de importação da biblioteca *PySide* diretamente;
- *end_qtapp*: executa aplicação *PySide* criada pelo método *begin_qtapp* (ou pela maneira nativa fornecida pela biblioteca *PySide*) de maneira alternativa, com mesma motivação do método anterior. Para que os elementos gráficos possam ter o seu fluxo de execução normal, é necessário executar essa aplicação que, em outras palavras, permite com que haja interação com o usuário e o tratamento adequado dos eventos decorrente dessa interação;
- *init_interactive_widget*: cria uma instância da classe *InteractiveWidget*, que será discutida na próxima seção. Este método recebe como parâmetro o *keyword argument*^{xxv} *'console'* que é um booleano para informar se a janela interativa de comandos do *Python* deve ser visualizada, onde o seu valor padrão é verdadeiro. O *keyword argument* *'namespace'* deve ser enviado por meio de um dicionário para mapear e permitir com que as variáveis contidas no mesmo sejam acessíveis na

^{xxv} Um *keyword argument* é um tipo de argumento que possui identificação na chamada de um método ou função no *Python*. Na existência simultânea de tipos comuns de argumentos, eles devem ser enviados no final da lista parâmetros.

janela interativa de comandos. Além disso, se este método já tiver sido executado por qualquer instância da classe *WidgetManager* será gerada uma exceção. Sendo assim, este método pode ser executado apenas uma vez e, conseqüentemente, apenas uma janela interativa deve existir, caso elas sejam inicializadas pela classe *WidgetManager*;

- *append_widget*: agrega um elemento gráfico do *PySide* na janela interativa. Deve ser enviado como argumento deste método o elemento gráfico que seja um objeto da classe *PySide.QtGui.QWidget* ou uma subclasse dela;
- *interactive_widget* (propriedade): retorna a instância da classe *InteractiveWidget*. Se o método *init_interactive_widget* não tiver sido executado nenhuma vez, então o valor de retorno será *None*^{xxvi}, representando a inexistência dessa instância.

iw_widgets (propriedade): retorna uma lista contendo todos os elementos gráficos adicionados através do método *append_widget*.

5.5 Widgets

Embora todo o conjunto de elementos gráficos da biblioteca *PySide* possa ser utilizado nos *scripts* de pós-processamento, alguns outros recursos tiveram a necessidade de ser implementados ou adaptados. Ferramentas para visualização de imagens ou gráficos certamente são necessários em aplicações de RM. Desta maneira, como não é intuito deste trabalho desenvolver estas ferramentas a partir fundamentos básicos, optou-se pela utilização de bibliotecas existentes.

Inicialmente, um conjunto relativamente considerável de bibliotecas foi observado. Entre elas, estava a biblioteca *Matplotlib*, já mencionada anteriormente e utilizada na visualização do sistema de aquisição do *ToRM Console*. Por ser bastante popular, a utilização desta biblioteca possui inúmeras vantagens como, por exemplo, conter uma documentação bem elaborada, ser bem testada, além da existência de uma comunidade relativamente grande de usuários e exemplos disponíveis. Entretanto, existem alguns problemas com a utilização desta biblioteca em dois quesitos: integração com *PySide* e desempenho. Ainda que seja possível integrá-la ao *PySide*, como foi mostrado no decorrer do desenvolvimento do *ToRM*

^{xxvi} *None* é um tipo nativo do *Python*, equivalente ao tipo *null* da linguagem C, por exemplo. Este tipo normalmente é utilizado para simbolizar um valor nulo.

Console, a utilização dos chamados *backends*, que são meios para permitir que esta biblioteca seja compatível com os diversos arcabouços de interface gráfica existentes, não é trivial e poderia ser fonte de confusão e, conseqüentemente, desmotivação para o programador dos *scripts* de pós-processamento, divergindo da proposta da criação desta API. Em relação ao desempenho, este problema pode não ser percebido com a utilização de baixa densidade de pontos plotados. No entanto, a falta de recursos gráficos que façam uso do potencial vetorial das placas de vídeo atuais como, por exemplo, o *OpenGL*, fazem com que esta biblioteca tenha baixo desempenho durante a visualização de imagens ou gráficos com altas densidades de pontos, apresentando uma experiência indesejada ao usuário.

Entre as demais opções, todas possuem um ou mais problemas semelhantes aos da *Matplotlib*, além de que algumas simplesmente não são compatíveis com o *PySide*. Por fim, uma biblioteca mostrou-se bastante interessante não apenas por ser compatível com o *PySide* e utilizar *OpenGL*, tendo um desempenho considerável, mas também por ter sido utilizada em uma aplicação que, em alguns aspectos, é muito semelhante ao contexto deste trabalho: a *PyQtGraph*. A *PyQtGraph* é uma biblioteca de código aberto que possui licença MIT e é voltada à aplicações científicas, contendo algumas funcionalidades relevantes além da visualização gráfica. Ela foi desenvolvida com o intuito de ser utilizada por um sistema de aquisição e análise de dados neurofisiológicos, o ACQ4. (40) As diversas ferramentas fornecidas por esta biblioteca serão apresentadas conforme o uso das mesmas surgirem durante as discussões subsequentes.

5.5.1 *InteractiveWidget*

Subclasse da *QMainWindow* da biblioteca *PySide*, a classe *InteractiveWidget* é responsável pela janela principal da biblioteca *ToRM Processing API*. O seu objetivo é agrupar todos os elementos gráficos gerados pelo *script* de pós-processamento, além de oferecer ferramentas que permitam ao usuário interagir com os resultados por meio de linha de comando do interpretador do *Python*. Este interpretador é acessado através de uma instância da classe já implementada pela biblioteca *PyQtGraph*, o *ConsoleWidget*. A figura 43 ilustra o funcionamento do *ConsoleWidget* através da janela interativa.

Figura 43 - *ConsoleWidget* sendo executado de dentro do *InteractiveWidget*. O usuário pode inserir comandos que obedeçam à sintaxe do *Python*.

Fonte: Elaborada pelo autor.

Através do submenu ‘*Save data*’ do menu ‘*File*’ é possível salvar os dados do tipo *Numpy Array* em um arquivo *HDF5*, semelhante ao *final_data.h5*, se eles forem registrados através do método *register_data* da classe *DataManager*, que foi discutido na subseção 5.4.2. Esta funcionalidade pode ser útil para gravar os dados processados pelo *script* de pós-processamento ou por comandos adicionais enviados pelo *ConsoleWidget*.

5.6 Exemplos de uso: utilização de protocolos e visualização de imagens e gráficos

O programador pode, a princípio, escolher qualquer ferramenta de visualização que seja compatível ao *PySide*, caso ele queira utilizar os recursos do *InteractiveWidget*. Entretanto, o conjunto de elementos gráficos oferecidos pela biblioteca *PyQtGraph* fornecem recursos suficientes para atender um conjunto vasto de aplicações. Nesta subseção serão apresentados alguns recursos desta biblioteca trabalhando em conjunto com as outras ferramentas do *ToRM Processing API* através de exemplos.

O primeiro exemplo é baseado em uma das mais básicas aplicações possíveis: gerar uma imagem aleatória e apresentá-la ao usuário. Os códigos serão apresentados na figura 44 de modo que a utilização da API fique clara para o leitor:

```

In [1]: import torm_processing_api as papi
import numpy as np
import pyqtgraph as pg

In [2]: wm = papi.WidgetManager()
wm.begin_qtapp()

In [3]: img = np.random.randn(256,256)
imgview = pg.ImageView()
imgview.setImage(img)

In [4]: #dicionário contendo variáveis a ser inicializado
namespace = {'img': img}
wm.init_interactive_widget(namespace = namespace)
wm.append_widget(imgview, "Image Test")

In [5]: wm.end_qtapp()

```

Figura 44 - Código Python necessário para visualização de uma imagem randômica de 256 x 256 pixels utilizando os recursos da *ToRM Processing API*, *PyQtGraph* e *Numpy*. A primeira etapa é a importação de todas as bibliotecas utilizadas (1). Em seguida, uma instância da classe *WidgetManager* é criada e seu respectivo método para inicialização de uma aplicação *PySide* é executado (2). Para que a imagem randômica seja criada, é feito uso da função *randn*, contida no pacote *numpy.random*, que retorna um objeto do tipo *Numpy Array* contendo um conjunto de pontos flutuantes, gerados a partir de uma distribuição normal, que será visualizado através do método *setImage* da instância da classe *ImageView* da biblioteca *PyQtGraph* (3). A janela interativa é inicializada e o espaço de variáveis que poderá ser utilizado pelo usuário é informado através do dicionário nomeado *namespace*. Além disso, o objeto *wm*, que representa o elemento gráfico contendo a imagem, é adicionado à janela interativa (4). Por fim, a aplicação do *PySide* é executada através do método *end_qtapp* da classe *WidgetManager*, permitindo com que o usuário possa interagir com a interface gráfica (5).

Fonte: Elaborada pelo autor.

Este exemplo está contido no módulo *image.py* do pacote *torm_processing_api.examples* e pode ser utilizado durante o aprendizado desta API. A figura 45 ilustra o resultado da execução deste *script*. Como é possível observar, o programador tem acesso à variável *'img'*, que contém os dados que representam a imagem aleatória. Outra característica que pode ser vista é a existência de um ajuste de limites de intensidades máxima e mínima da imagem, contendo um histograma que, neste caso, revela a distribuição gaussiana utilizada para gerar os pontos da imagem.

O próximo exemplo, também contido no pacote *torm_processing_api.examples*, visa demonstrar a utilização das classes *ProtocolManager*, *DataManager* e *PathManager*. Apenas o trecho de código correspondente à utilização destas classes será mostrado na figura 46, tendo em vista a semelhança entre as outras partes com as que estão contidas no exemplo anterior.

Figura 45 - Resultado do código executado na figura 44. A classe *ImageView* da biblioteca *PyQtGraph* permite a visualização da imagem contendo ferramentas básicas como, por exemplo, *zoom* (através da rolagem do *mouse*), controle de intensidade máxima e mínima.

Fonte: Elaborada pelo autor.

```
In [1]: pathm = papi.PathManager("data/SE_Mul")

pm = papi.ProtocolManager()
pm.open_protocols(acq_path = pathm.acq_protocol_path,
                  exec_path = pathm.exec_protocol_path)

dm = papi.DataManager()
data = dm.read_data(pathm.data_path,
                    pathm.acq_protocol_path)
```

Figura 46 - Utilização das classes *ProtocolManager*, *DataManager* e *PathManager*. Um conjunto de dados sintetizados deve ser colocado dentro do endereço “data/SE_Mul”, relativo ao *script* executado, que deve conter a mesma estrutura mencionada na seção 5.3. A classe *PathManager* é utilizada para se obter os caminhos dos protocolos de execução e aquisição e dos dados. Estas informações são utilizadas pelos métodos *open_protocols* da classe *ProtocolManager* e *read_data* da classe *DataManager*.

Fonte: Elaborada pelo autor.

O resultado da execução do *script* que compõe este exemplo é mostrado na figura 47. Parte do trecho de código omitido é responsável por instanciar o objeto da classe *PlotWidget*, da biblioteca *PyQtGraph*, que será responsável pela visualização gráfica de parte dos dados carregados.

Figura 47 - Demonstração da leitura de dados e protocolos através da biblioteca *ToRM Processing API*. Os dados visualizados foram sintetizados com a mesma técnica que será mencionada no capítulo 6. São apresentados no gráfico a parte real (azul), imaginária (verde) e módulo (vermelho) de parte destes dados. Os protocolos podem ser acessados pela linha de comando por meio do objeto *pm* da classe *ProtocolManager*. Alguns métodos desta classe, já discutidos neste capítulo, são demonstrados.

Fonte: Elaborada pelo autor.

Os elementos gráficos adicionados através do método *append_widget* da classe *InteractiveWidget* são colocados em uma instância da classe *QDockWidget*, da biblioteca *PySide*. Sendo assim, quando mais de um elemento gráfico são adicionados, os seguintes comportamentos são possíveis, conforme mostrado na figura 48:

Figura 48 - Comportamento de múltiplas janelas inseridas na janela interativa.

Fonte: Elaborada pelo autor.

6 RESULTADOS

6.1 Operação com dados sintetizados

Alguns dos testes realizados com o espectrômetro foram feitos através do uso de dados sintetizados. Estes dados são obtidos da mesma forma em que acontece durante a realização de um experimento real. Entretanto, o sinal é gerado através do acoplamento de um sintetizador digital desenvolvido pelo grupo ao receptor do espectrômetro. Este sintetizador é implementado através de uma FPGA e possui algumas funcionalidades que garantem o sincronismo com a execução do método, de modo que os dados sejam gerados e adquiridos no intervalo de tempo desejado. A figura 49 ilustra o sistema em operação no modo de testes.

Figura 49 - Sistema em funcionamento no modo de testes através do uso de sintetizador de sinal. À esquerda o *ToRM Console* está sendo executado para aquisição do sinal. À direita o osciloscópio mostra parte das formas de ondas geradas pelo método em execução e abaixo dele o espectrômetro acoplado ao sintetizador é mostrado.

Fonte: Documentação interna do CIERMag.

As primeiras funcionalidades testadas com o uso desta abordagem foram os filtros e o uso de médias para redução de banda e remoção de ruído. Um método com apenas *loops* de médias foi utilizado. A figura 50 mostra a primeira amostra de dados recebida que contém 8192 pontos complexos em uma banda de 1 MHz, no intervalo de frequências entre -500 kHz e $+500$ kHz, sem nenhum processamento realizado pelo *ToRM Console*. Em seguida, a

figura 51 ilustra a aplicação do sistema de filtragem com frequência de corte de 31,250 kHz (banda passante entre -31,250 kHz e +31,250 kHz) com decimação para 512 pontos complexos (fator de decimação 16). Como é possível observar, as componentes de alta frequência do ruído foram removidas. Por fim, ao final da aplicação de 128 médias, o ruído contido dentro da região espectral de interesse foi removido, conforme mostrado na figura 52.

Figura 50 - Eco sintetizado com ruído em banda de 1 MHz. O conjunto amostral possui 8192 pontos coletados em fase e quadratura (representados em números complexos). A parte real (azul), imaginária (verde) e o módulo correspondente (vermelho) são apresentados simultaneamente.

Fonte: Elaborada pelo autor.

Figura 51 - Conjunto de dados mostrado na figura 50 após a aplicação de filtro passa-baixa FIR com banda passante entre -31,250 kHz e 31,250 kHz. O ruído de alta frequência foi reduzido.

Fonte: Elaborada pelo autor.

Figura 52 - Continuação do mesmo experimento mostrado na figura 51 e na figura 52 após a aplicação de 128 médias. O ruído presente na região espectral com módulo abaixo da frequência de corte também é eliminado.

Fonte: Elaborada pelo autor.

Os métodos ‘SE_Mul’, utilizado para obtenção de múltiplos planos através do uso de ecos de spin; (41) e ‘TurboSE’, baseado no método originalmente descrito em (42), foram utilizados também com dados sintetizados para testes no sistema de organização de dados.

O método ‘SE_Mul’ possui, do mais externo ao mais interno, os seguintes *loops*: ‘PhEnc’, do tipo ‘PhEI’; ‘Average’, do tipo de ‘Average’ que, portanto, é utilizado para médias; e ‘Slice’ do tipo ‘Slice’, utilizado para obtenção de múltiplas fatias. A matriz de dados final, desconsiderando a dimensão do conjunto de dados, possui apenas duas dimensões, uma vez que a dimensão de médias é colapsada.

Figura 53 – Método ‘SE_Mul’ em execução. A dimensão de médias é colapsada, resultando apenas duas dimensões no conjunto de dados armazenados.

Fonte: Elaborada pelo autor.

O método ‘Turbo_SE’ possui, do mais externo ao mais interno, os *loops* do tipo: ‘Split’, ‘PhEI’, ‘Average’, ‘Slice’, ‘EchoTrainPhEI’. Sendo assim, através do conjunto de dados contido no arquivo final armazenado, foi possível observar a redução do número de dimensionalidades para apenas duas dimensões. A dimensão de ‘Split’ aumenta o número efetivo de *slices* e, em seguida, é truncada. A dimensão de médias também é truncada. E, por fim, a dimensão ‘EchoTrainPhEI’ recebe o truncamento, de maneira que cada eco contido nesta dimensão se transforme em um valor para codificação de fase a mais.

Figura 54 - Método ‘Turbo_SE’ em execução. Apenas as dimensões relativas à codificação de fase e *slice* continuam após o procedimento de organização de dados.

Fonte: Elaborada pelo autor.

Conforme mencionado no capítulo 3, o magneto disponível para testes com experimentos reais até a data de depósito desta dissertação de mestrado possui algumas limitações. Entre elas, a intensidade do campo magnético (0.5 T) dificulta a utilização de técnicas de espectroscopia. Além disso, por ser um magneto permanente, os efeitos de corrente de Foucault observados durante a utilização de bobinas de gradiente inviabilizaram a aquisição de imagens.

Estes problemas foram previstos durante o projeto desta instrumentação para que a mesma pudesse ser utilizada para testar algumas funcionalidades de correção do *hardware* do

espectrômetro. Entretanto, o tempo disponível para conclusão deste trabalho foi insuficiente para que o grupo pudesse realizar as devidas correções nos métodos e nos componentes de *hardware* que permitissem a obtenção de imagens.

Desta forma, para mostrar o funcionamento do *ToRM Console* em conjunto com as ferramentas oriundas da *ToRM Processing API*, um conjunto de dados foi simulado via *software* para simulação de aquisição de imagens. A partir da transformada de Fourier bidimensional de uma imagem de 256x256 *pixels* contendo o logo do CIERMag^{xxvii}, foram criados arquivos idênticos aos que são criados pelo *ACQServer* contendo as respectivas linhas do espaço k.

Os dados foram sintetizados de maneira a simular um experimento utilizando o método ‘SE_Mul’ com 256 codificações de fase e 256 amostragens em cada uma delas. Apenas um *slice* e uma média foram utilizados. Além disso, o sistema de filtragem não foi utilizado. Todas as outras etapas foram percorridas normalmente, forçando apenas o *ToRM Console* utilizar outro diretório durante a aquisição dos dados.

Um *script* de pós-processamento foi desenvolvido para realizar leitura dos dados e visualização. Durante a visualização são mostrados os dados adquiridos (espaço k) e a imagem obtida através da transformada bidimensional inversa de Fourier. A figura 55 mostra o *ToRM Console* durante a aquisição dos dados, enquanto que a figura 56 mostra o resultado do *script* de pós-processamento executado automaticamente após o término da aquisição.

^{xxvii} Não foram utilizadas imagens obtidas através de experimentos reais de ressonância magnética em outros equipamentos propositalmente, de maneira a enfatizar o uso de dados sintetizados.

Figura 55 - Execução do método 'SE_Mul' com a utilização de dados sintetizados via *software* através de imagem do logo do CIERMag. O dado visualizado representa uma linha no espaço k.

Fonte: Elaborada pelo autor.

Figura 56 - *Script* de pós-processamento para o método "SE_Mul" no mesmo experimento mostrado na figura 55. A partir da aplicação da transformada inversa de Fourier bidimensional sobre o conjunto de dados obtidos (espaço k, à esquerda) foi possível reconstruir a imagem (à direita).

Fonte: Elaborada pelo autor.

6.2 Experimentos de relaxometria: CPMG para medidas de T_2

O método de sequência de pulsos Carr-Purcell-Meiboom-Gill (CPMG) pode ser utilizado para o cálculo do tempo de relaxação da componente transversal da magnetização, T_2 . (43-44) Um método que recebe o nome ‘CPMG_E’ foi implementado pelo grupo para ser utilizado no espectrômetro CIERMag. Através da instrumentação disponível, discutida anteriormente, serão apresentados os resultados da utilização deste método. A figura 57 mostra o equipamento utilizado.

Figura 57 - Espectrômetro com magneto permanente de 0.5 T utilizado para a realização de testes de *hardware* e *software* do sistema. A foto à direita mostra o *probe*, onde é possível colocar amostras no interior de tubos de ensaio.

Fonte: Documentação interna do CIERMag.

Ao invés de apenas demonstrar valores de tempos de relaxação transversal para diferentes amostras, será mostrado o comportamento da taxa de relaxação transversal, $R_2 = 1/T_2$, de amostras contendo sulfato de cobre (CuSO_4) diluídas em água destilada em função da variação da concentração. Este comportamento é conhecido na literatura. (45)

O T_2 pode ser medido através da curva formada pela obtenção dos pontos máximos dos ecos obtidos na execução do método. Para isso, foi desenvolvido um *script* de pós-processamento utilizando os recursos do *ToRM Processing API*. Este *script* é responsável por obter a curva de decaimento e, através da função *curve_fit* do pacote *scipy.optimize*, que utiliza o algoritmo de Levenberg-Marquardt, encontrar os parâmetros que constituem a função com melhor ajuste de decaimento exponencial ou multi-exponencial^{xxviii}, incluindo os tempos de relaxação transversal. (46-47) Além disso, a curva que compõe o melhor ajuste exponencial e os pontos máximos dos ecos é visualizada.

^{xxviii} Em função de restrições matemáticas, é possível obter os valores dos parâmetros de, no máximo, três componentes exponenciais distintas através deste método.

As amostras foram preparadas em um volume de 50 ml com as concentrações mostradas na tabela abaixo:

Tabela 2 - Massa e concentração das amostras (A, B, C e D) de CuSO_4 utilizadas no experimento.

Amostra	Massa de CuSO_4 (g)	Concentração (g/l)
A	0,004	0,08
B	0,006	0,12
C	0,027	0,54
D	0,050	1,00

Fonte: Elaborada pelo autor.

As figuras 59, 59 e 60 mostram o *ToRM Console* em execução durante a etapa de obtenção do protocolo mais adequado para a realização dos experimentos. Após esta etapa, o protocolo definido foi: 350 ecos, 64 pontos por eco (32 após filtragem), 8 médias, tempo de repetição (T_r) de 6 s, tempo entre ecos (T_E) de 10 ms e frequência de corte de 250 KHz.

A sequência ‘CPMG_E’ possui dois *loops*: um do tipo ‘Average’ (externo) e outro do tipo ‘EchoTrain’ (interno). Sendo assim, os dados são somados e normalizados no fim da execução e é possível visualizar simultaneamente o decaimento dos múltiplos ecos adquiridos. É possível observar nestas figuras a mudança do tempo de relaxação em cada um dos casos com a variação da concentração de sulfato de cobre.

Figura 58 - *ToRM Console* executando o método ‘CPMG_E’ com a amostra D, que possui concentração de 1g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.

Fonte: Elaborada pelo autor.

Figura 59 - *ToRM Console* executando o método ‘CPMG_E’ com a amostra C, que possui concentração de 0,54 g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.

Fonte: Elaborada pelo autor.

Figura 60 - *ToRM Console* executando o método 'CPMG_E' com a amostra A, que possui concentração de 0,08 g/l. A visualização está sendo feita no domínio do tempo e com o botão de médias desligado, de maneira a mostrar o último dado adquirido no instante em que a figura foi feita.

Fonte: Elaborada pelo autor.

Inicialmente, foi utilizado um modelo com apenas uma exponencial, sabendo que apenas uma componente deveria existir neste tipo de amostra. Entretanto, ao observar o comportamento do ajuste exponencial foi possível concluir que este modelo não era o mais apropriado. Com a utilização de um modelo com duas exponenciais foi observado, em todas as amostras, a existência de uma componente adicional com valores de tempo de relaxação transversal entre 70 e 90 ms, aproximadamente. Desta vez, a curva contendo o melhor ajuste foi mais apropriada ao conjunto amostral, conforme é possível observar nas figuras 61, 62, 63 e 64. Esta anomalia pode ter sido causada pela existência de impurezas ou efeitos de difusão.

Foram realizadas três medidas para cada amostra e o valor médio do tempo de relaxação transversal da componente de interesse, ou seja, relativa à solução de sulfato de cobre, foi utilizado para calcular a taxa de relaxação, R_2 . A tabela 3 mostra os resultados obtidos.

Tabela 3 - Valores de tempos de relaxação obtidos para cálculo da taxa de relaxação calculada a partir do valor de T_2 médio.

Amostra	T_2 (ms)			T_2 Médio (ms)	R_2 (s^{-1})
A	1012	1007	974	998	1,00
B	962	957	952	957	1,04
C	706	654	647	669	1,50
D	390	385	373	383	2,61

Fonte: Elaborada pelo autor.

Figura 61 - *Script* de pós-processamento em execução com auxílio do *ToRM Processing API* para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência ‘CPMG_E’ com a amostra A. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.

Fonte: Elaborada pelo autor.

Figura 62 - *Script* de pós-processamento em execução com auxílio do *ToRM Processing API* para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência ‘CPMG_E’ com a amostra B. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.

Fonte: Elaborada pelo autor.

Figura 63 - *Script* de pós-processamento em execução com auxílio do *ToRM Processing API* para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência ‘CPMG_E’ com a amostra C. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.

Fonte: Elaborada pelo autor.

Figura 64 - *Script* de pós-processamento em execução com auxílio do *ToRM Processing API* para obtenção dos pontos máximos dos ecos e cálculo da melhor curva exponencial da sequência ‘CPMG_E’ com a amostra D. O modelo com duas exponenciais foi utilizado para ajuste da melhor curva.

Fonte: Elaborada pelo autor.

Figura 65 - Taxa de relação transversal, $R_2 = 1/T_2$, em função da concentração de CuSO_4 , apresentando comportamento linear, conforme previsto pela literatura.

Fonte: Elaborada pelo autor.

A figura 65 mostra a curva de taxa de relaxação transversal, R_2 , em função da concentração de CuSO_4 , confirmando o comportamento linear esperado conforme a literatura citada anteriormente.

7 CONCLUSÕES, LIMITAÇÕES E TRABALHOS FUTUROS

7.1 Conclusões

Os resultados obtidos através da simulação de experimentos com o uso de dados sintetizados demonstra a capacidade do *ToRM Console* de executar tipos distintos de métodos de sequência de pulsos desenvolvidos através dos editores que compõem o sistema do CIERMag.

Além disso, a realização do experimento para estudo do comportamento do tempo de relaxação transversal de soluções de sulfato de cobre em função da variação da concentração utilizando o método CPMG mostra que o subsistema do *ToRM Console*, em conjunto com as ferramentas providas do *ToRM Processing API* são suficientes para que o espectrômetro do CIERMag possa atuar, neste momento, como um Relaxômetro de fácil operação.

A maneira com que os dados são organizados automaticamente através do uso de tipos especiais de *loops*, discutida em detalhes na seção 4.8, permite com que o programador do método de sequências de pulso possa lidar com este tipo de tratamento de maneira transparente. Este tipo de abordagem reduz parte do tempo necessário para o desenvolvimento de um novo método e reduz o número de procedimentos necessários para reconstrução dos dados no *script* de pós-processamento.

As tecnologias elegidas, por serem atuais e conterem inúmeros recursos que permitem com que operações de processamento de sinais e visualização possam ser feitas em tempo relativamente baixo, sem perder as vantagens que uma linguagem de propósito geral possui, mostram-se adequadas para cumprir a proposta de flexibilidade.

A documentação que foi criada com o uso de ferramentas que automatizam parte deste processo permitirá com que outros desenvolvedores e pesquisadores possam entender o funcionamento interno dos subsistemas desenvolvidos. Esta característica é importante especialmente porque a continuidade deste trabalho, por meio da criação e adaptação das funcionalidades ainda necessárias no sistema como um todo, está prevista pelo grupo.

As novas necessidades, bem como o surgimento de novas tecnologias que são utilizadas para compor os equipamentos e *software* de RM são responsáveis pelo fenômeno de obsolescência dos sistemas, que é muito comum na área. De modo geral, este trabalho representa parte da contínua evolução do grupo em relação ao desenvolvimento de

instrumentação para ressonância magnética nuclear, podendo ser considerada como resposta a este fenômeno.

7.2 Limitações e trabalhos futuros

Este trabalho, por se tratar principalmente do desenvolvimento de *software* para operação de um equipamento de RM, cria diversas oportunidades para trabalhos futuros, tanto na implementação das funcionalidades que ainda são necessárias para o sistema quanto, no aprimoramento das que já estão implementadas, em função das limitações que não puderam ser sanadas por questões de disponibilidade de tempo.

Em relação ao gerenciamento de experimentos, ainda é necessário idealizar e desenvolver ferramentas para o controle das amostras utilizadas, bem como prover um sistema de visualização *off-line* dos dados obtidos no contexto do *ToRM Console*, que poderia utilizar os recursos do *ToRM Processing API* para realização de leitura, visualização e processamentos dos dados.

A execução de experimentos para obtenção de imagens necessita de uma etapa denominada *prescrição*. Nesta etapa uma imagem preliminar é feita para que o operador possa selecionar a região de interesse, bem como as fatias que são feitas em outro *scan*. A etapa de prescrição pode ser implementada através da alimentação do protocolo de execução do *scan* subsequente. Muitas das ferramentas de visualização necessárias para o desenvolvimento desta funcionalidade já estão disponíveis, conforme visto no capítulo 5. Além disso, o armazenamento das imagens deve ser feito em formatos mais apropriados, como é o caso do formato sugerido pelo *Digital Imaging and Communications in Medicine* (DICOM). (48)

Para que o sistema de usuários seja realmente seguro, o *ACQServer* deve possuir ferramentas de autenticação que utilize a mesma base de dados utilizada pelo *ToRM Console* e procedimentos adicionais de autenticação entre estes dois subsistemas devem ser incluídos.

Em relação ao sistema de aquisição, o *ToRM Console* ainda deve ser adaptado para a obtenção de dados por meio de múltiplos receptores, tendo em vista que todos os testes e experimentos foram realizados com um único receptor. Além disso, o sistema de filtragem continua apresentando problemas de desempenho quando filtros de ordem alta são utilizados em experimentos de tempo de repetição curto. Este problema pode ser contornado com a

aplicação de múltiplos filtros de ordem baixa em cascata ou então por meio de processamento utilizando GPU.

Por fim, outros dispositivos que farão parte futuramente do espectrômetro (controle da cama do paciente, *shimming*, controle de temperatura, *etc.*) devem ser controlados pelo *ToRM Console*.

REFERÊNCIAS*

- 1 ANDRADE, E. I. G. et al. Analysis of the knowledge evolution and the diffusion and incorporation of the magnetic resonance technology in Brazil. In: ANNUAL MEETING HEALTH TECHNOLOGY ASSESSMENT INTERNATIONAL (HTAi), 4., 2007, Barcelona. **Abstracts** ...Barcelona: [s.n.], 2007. p. 271–271.
- 2 JOB, C.; PEARSON, R. M.; BROWN, M. F. A personal computer-based nuclear magnetic resonance spectrometer. **Review of Scientific Instruments**, v. 65, n. 11, p. 3354, 1994.
- 3 STALLINGS, W. **Computer organization and structure: designing for performance**. 6th ed. Michigan: Prentice Hall, 2003. 815 p.
- 4 HUETTEL, S. A.; SONG, A. W.; MCCARTHY, G. **Functional magnetic resonance imaging**. 2nd ed. Sunderland, Mass: Sinauer Associates, 2008.
- 5 ESTRIN, G. Reconfigurable computer origins: the UCLA fixed-plus-variable (F+V) structure computer. **IEEE Annals of the History of Computing**, v. 24, n. 4, p. 3-9, 2002.
- 6 ROSE, J. et al. Architecture of field-programmable gate arrays. **Proceedings of the IEEE**, v. 81, n. 7, 1993. Disponível em: <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=231340>>. Acesso em: 15 jan. 2014.
- 7 STANG, P. P. et al. Medusa: a scalable MR console using USB. **IEEE Transactions on Medical Imaging**, v. 31, n. 2, p. 370-379, Fev. 2012.
- 8 PYTHON Software Foundation. **Python programming language** – official website. Disponível em: <www.python.org/>. Acesso em: 14 jan. 2014.
- 9 OLIPHANT, T. E. Python for scientific computing. **Computing in Science & Engineering**, v. 9, n. 3, p. 10-20, 2007.
- 10 HUNTER, J. D. Matplotlib: a 2D graphics environment. **Computing in Science & Engineering**, v. 9, n. 3, p. 90-95, 2007.

* De acordo com a Associação Brasileira de Normas Técnicas NBR 6023.

11 THE MATHWORKS Inc. **MATLAB**: the language of technical computing. Disponível em: <<http://www.mathworks.com/products/matlab/>>. Acesso em: 14 jan. 2014.

12 THE GNOME project and PyGTK team. **PyGTK**. Disponível em: <<http://www.pygtk.org/>>. Acesso em: 14 jan. 2014.

13 CHAUDHARY, B. **Tkinter GUI application development hotshot**. Birmingham, UK: Packt Publishing, 2013.

14 RIVERBANK computing limited. Disponível em: <www.riverbankcomputing.co.uk/software/pyqt/intro>. Acesso em: 14 jan. 2014.

15 LOGANATHAN, V. **PySide GUI application development**. Birmingham: Packt Publishing, 2013.

16 KDE. Disponível em: <www.kde.org>. Acesso em: 14 jan. 2014.

17 BLANCHETTE, J.; SUMMERFIELD, M. **C++ GUI programming with Qt 4**. [s.l.] Prentice Hall, 2006. 560 p.

18 GERLACH, W.; STERN, O. Das magnetische moment des silberatoms. **Zeitschrift fur Physik**, v. 9, n. 1, p. 353–355, dez. 1922.

19 HAACKER, E. M. et al. **Magnetic resonance imaging**: physical principles and sequence design. 2nd ed. New York: John Wiley & Sons Inc., 2014.

20 BERSOHN, M.; BAIRD, J. C. **An introduction to electron paramagnetic resonance**. New York: W.A. Benjamin, 1966. 272 p.

21 LANDÉ, A. Nuclear Magnetic Moments and Their Origin. **Physical Review**, v. 46, n. 6, p. 477–480, Sept. 1934.

22 COELHO, F. B. et al. Domain specific language for Magnetic Resonance Sequence programming on a digital spectrometer. In: WORKSHOP CInAPCe, 5., 2011, São Paulo. **Anais...** São Paulo:[s.n.], 2011.

23 THE ECLIPSE foundation. **Eclipse**: the eclipse foundation open source community website. Disponível em: <www.eclipse.org>. Acesso em: 14 jan. 2014.

24 THE HDF group. **HDF Group - HDF5**. Disponível em: <www.hdfgroup.org/HDF5/>. Acesso em: 14 jan. 2014.

25 SPI. Debian - the universal operating system. Disponível em: <www.debian.org>. Acesso em: 14 jan. 2014.

26 CANONICAL Ltd. **Meet Ubuntu**. Disponível em: <www.ubuntu.com/desktop>. Acesso em: 14 jan. 2014.

27 QT Project. Disponível em: <qt-project.org/doc/qt-4.8/designer-manual.html>. Acesso em: 14 jan. 2014.

28 DUSTMAN, A. **MySQLdb user's guide**. Disponível em: <<http://mysql-python.sourceforge.net/MySQLdb.html>>. Acesso em: 14 jan. 2014.

29 JSON. Disponível em: <www.json.org>. Acesso em: 14 jan. 2014.

30 W3C. **Extensible markup language (XML)**. Disponível em: <www.w3.org/XML/>. Acesso em: 14 jan. 2014.

31 THE PYTHON software foundation. **20.8. ftplib — FTP protocol client — Python v2.7.6 documentation**. Disponível em: <docs.python.org/2/library/ftplib.html>. Acesso em: 14 jan. 2014.

32 THE PYTHON software foundation. **16.6. multiprocessing — process-based “threading” interface — Python v2.7.6 documentation**. Disponível em: <docs.python.org/2/library/multiprocessing.html>. Acesso em: 14 jan. 2014.

33 MATTSON, T.; SANDERS, B. A.; L. MASSINGILL, B. **Patterns for parallel programming**. Boston: Addison-Wesley Professional, 2004.

34 BRIGHAM, E. O. **The fast Fourier transform and its applications**. Englewood Cliff: Prentice-Hall, 1988.

35 HARRIS, F. J. On the use of windows for harmonic analysis with the discrete Fourier transform. **Proceedings of the IEEE**, v. 66, n. 1, p. 51–83, 1978.

36 THE HDF GROUP. The HDF group - information, support, and software. Disponível em: <www.hdfgroup.org/products/java/hdf-java-html/hdfview/index.html>. Acesso em: 14 jan. 2014.

37 BRANDL, G. **Overview — Sphinx 1.2.1 documentation**. Disponível em: <<http://sphinx-doc.org/>>. Acesso em: 14 jan. 2014.

38 COLLETTE, A. et al. **HDF5 for Python**. Disponível em: <<http://www.h5py.org/>>. Acesso em: 14 jan. 2014.

39 PYTHON software foundation. **Distributing python modules**: Python v2.7.6 documentation. Disponível em: <docs.python.org/2/distutils/>. Acesso em: 14 jan. 2014.

40 CAMPAGNOLA, L.; KRATZ, M. B.; MANIS, P. B. ACQ4: an open-source software platform for data acquisition and analysis in neurophysiology research. **Frontiers in Neuroinformatics**, v. 8, n. 3, 2014. doi: 10.3389/fninf.2014.00003.

41 MANSFIELD, P. Multi-planar image formation using NMR spin echoes. **Journal of Physics C: solid state physics**, v. 10, n. 3, p. L55–L58, 14 fev. 1977.

42 HENNIG, J.; NAUERTH, A.; FRIEDBURG, H. RARE imaging: a fast imaging method for clinical MR. **Magnetic Resonance in Medicine**, v. 3, n. 6, p. 823–833, 18 Dec. 1986.

43 CARR, H.; PURCELL, E. Effects of diffusion on free precession in nuclear magnetic resonance experiments. **Physical Review**, v. 94, n. 3, p. 630–638, May. 1954.

44 MEIBOOM, S.; GILL, D. Modified spin-echo method for measuring nuclear relaxation times. **Review of Scientific Instruments**, v. 29, n. 8, p. 688-91, 1958.

45 PAIVA, F. F. **Metodologia de imagens de NMR que utiliza um único pulso adiabático de inversão**. 2004. 116 p. Dissertação (Mestrado) - Instituto de Física de São Carlos, Universidade São Paulo, São Carlos, 2004.

46 LEVENBERG, K. A method for the solution of certain problems in least squares. **Quarterly of Applied Mathematics**, v. 2, p. 164–168, 1944.

47 MARQUARDT, D. An Algorithm for least-squares estimation of nonlinear parameters. **SIAM Journal on Applied Mathematics**, v. 2, p. 431–441, 1963.

48 MILDENBERGER, P.; EICHELBERG, M.; MARTIN, E. Introduction to the DICOM standard. **European Radiology**, v. 12, n. 4, p. 920–7, Apr. 2002.