DETECÇÃO DE MARCADORES BIOLÓGICOS MAGNÉTICOS

Aluno: Jan Krueger Siqueira
Orientador: Antonio Carlos Oliveira Bruno

Introdução

Ensaios imunológicos em amostras biológicas consistem em métodos que medem a reação antígeno-anticorpo através de um marcador ligado ao anticorpo. Recentemente, métodos magnéticos de detecção têm sido aplicados a estes ensaios através da utilização de marcadores contendo nanopartículas magnéticas em seu núcleo, na tentativa de aumentar a sensibilidade do ensaio. Isto pode levar a um diagnóstico precoce de determinadas patologias como tumores, doenças auto-imunes, etc. Porém é necessário, antes de tudo, caracterizar cada tipo de partícula e avaliar assim qual a melhor utilização para cada uma.

Objetivos

Calibração e configuração de um sistema de medição magnética utilizando um eletroímã, alimentado por uma fonte de corrente de bipolar. Além de outras aplicações, o sistema levantará curvas de magnetização de amostras ferro-magnéticas e pára-magnéticas e assim obter suas propriedades. Para tal, foram utilizados os softwares LabView, que cuida da comunicação entre o computador e os equipamentos, e MatLab, que processa os pontos medidos e gera as curvas desejadas.

Metodologia

A fim de levantarmos uma curva de magnetização, é necessário variar o campo externo H de zero a um valor máximo positivo, depois dali a um máximo negativo, e enfim de volta ao máximo positivo, medindo a resposta da densidade de fluxo magnético B da amostra no percurso (Figura 1). Alternativamente, o gráfico pode apresentar o momento de dipolo m normalizado pela massa da amostra no lugar de B, o que é mais comum nas especificações de certos materiais.

Sendo módulo e sentido do campo de um eletroímã [1] proporcionais a módulo e sentido da corrente que por ele passa, podemos controlar o campo externo manipulando a corrente (Figura 2). Portanto, o primeiro passo foi a configuração de uma nova fonte [2], esta especial por ser capaz de receber do usuário também o sentido de corrente desejado. Consultando a documentação, descobriu-se como controlá-la pelo computador via conexão serial, usando um conjunto básico de instruções. Com isso, já foi possível gerar então o campo variável controlado.

Resolvida a magnetização, passou-se ao sistema de medição da densidade de fluxo magnético da amostra. Aproveitando os conhecimentos e resultados da Iniciação Científica do ano anterior, utilizou-se um magnetômetro de efeito Hall [3], o qual também seria controlado por computador (porém via GPIB e conversores AD).

Para esse tipo de medição a técnica mais utilizada é a da extração. Ou seja, optou-se por realizar trajetórias retilíneas com a amostra, através de um atuador [4]; em seguida, compara-se a forma da curva B x posição com o modelo teórico de um dipolo-magnético (método também já utilizado na Iniciação Científica do ano anterior). Outra decisão foi a de medir a componente de B perpendicular a H, pois se notou que assim ruídos e interferências eram menores. Veja a figura 3.

Desta forma, o programa desenvolvido em LabView segue os seguintes passos: solicita um valor de corrente (que gera o campo H) e desloca a amostra ao longo de uma

distância (em torno de 1.5 cm), medindo a densidade de fluxo B gerada por ela em cada ponto do percurso (Figura 4). Tal processo pode ser repetido outras vezes para obtermos uma curva média, visando diminuir erros experimentais. Em seguida, um novo valor de corrente é solicitado (movendo um passo na curva de magnetização) e se reinicia o percurso.

Vale destacar que o material medido não pode sair da área submetida ao campo externo uniforme, pois isso causaria pequenos ciclos de histerese que prejudicariam a medição.

No fim, todas as curvas de medição estão salvas num arquivo texto. Desse ponto, uma rotina de instruções do MatLab analisa cada uma delas e tenta encontrar, pelo modelo teórico, o momento de dipolo m que melhor descreve o que foi observado. Feito isso, temos uma curva de magnetização m x H.

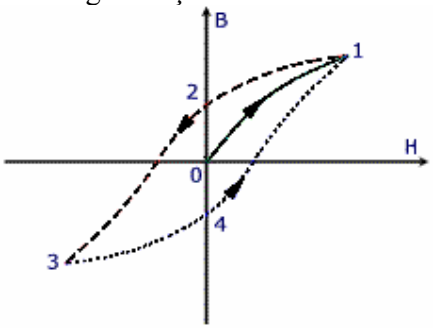


Figura 1: exemplo de curva de magnetização B x H

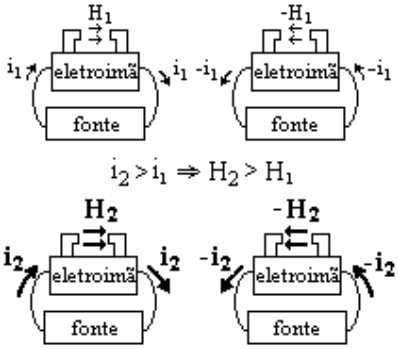


Figura 2: controle do eletroímã pela fonte de corrente

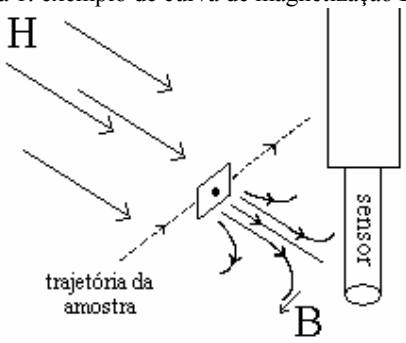


Figura 3: medição de B (perpendicular a H) da amostra, ao longo da trajetória

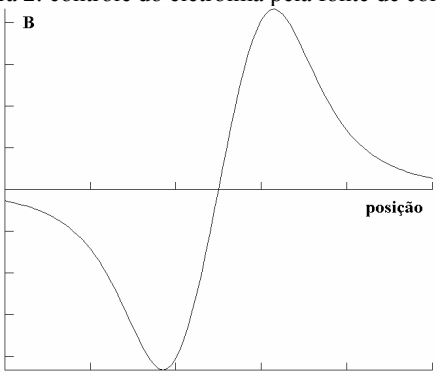


Figura 4: B ao longo da trajetória

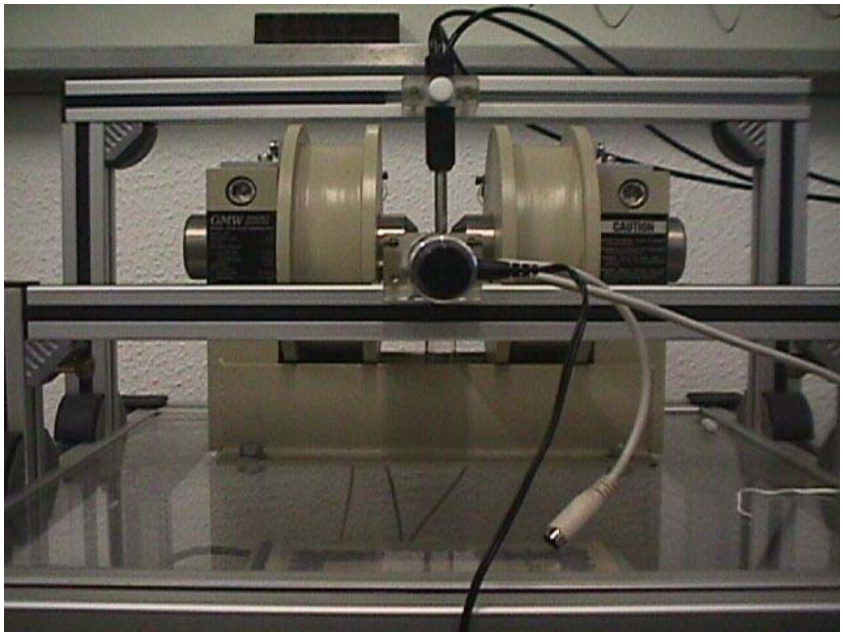
Conclusões

Este sistema de medição apresentou diversas dificuldades de ajuste, principalmente em função de ruídos e interferências. Por isso, foram feitos testes iniciais com materiais altamente magnetizáveis, a fim de que o valor de B medido fosse alto o bastante para resultar em curvas de boa análise. Além disso, filtros passa baixa freqüência foram utilizados.

Solucionados todos os problemas, o sistema já se mostra pronto para medir e caracterizar as diversas partículas nanomagnéticas.

Equipamento usado

[1] GMW Electromagnet, modelo 3470:



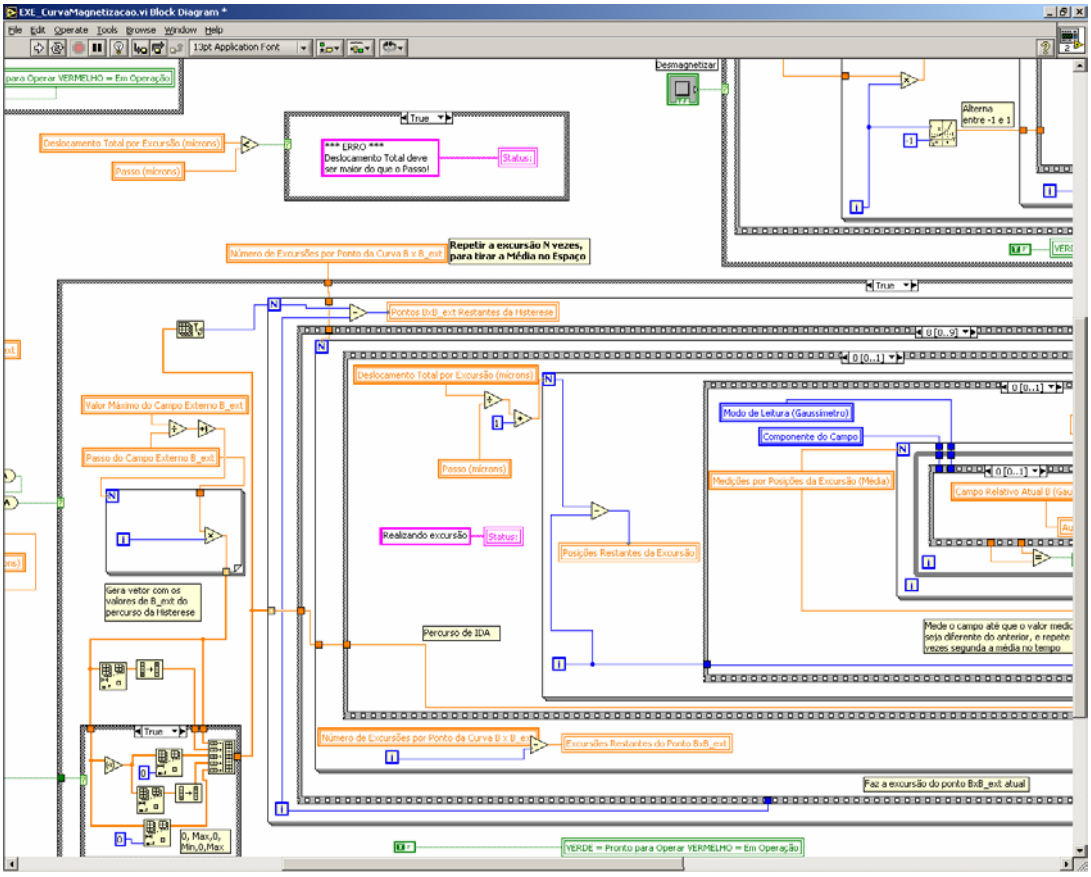
[2] Kepco Bipolar Operation Power Supply/Amplifier:

[3] F.W. Bell Series 9950 Gauss/Teslameter:

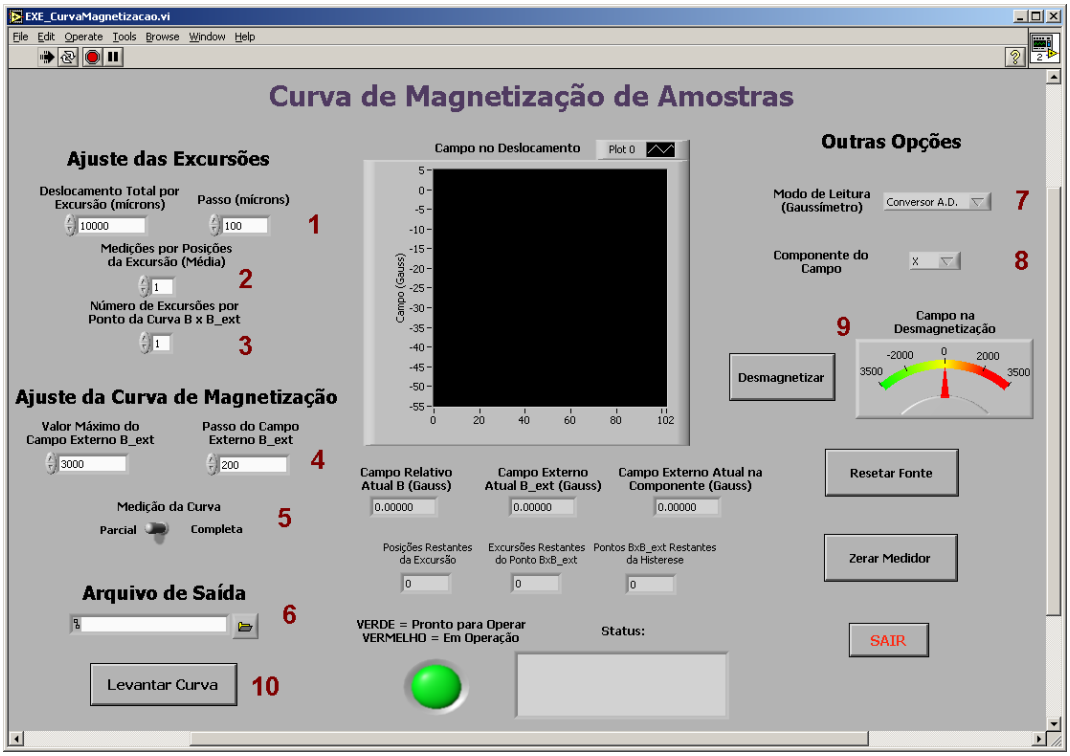
[4] Zaber Precision Linear Actuator, modelo T-LA60A:

Apêndice 1: **Detalhes do programa em LabView**

Como a programação em LabView é gráfica, torna-se complicado expô-la em detalhes. Abaixo, encontra-se uma captura de tela que mostra parte do que está por trás do funcionamento do painel, seguido então de como utilizá-lo.



OBS: para facilitar a nomenclatura, a "densidade de fluxo magnético" será referida como "campo".



A fim de se configurar a medição da magnetização da amostra, deve-se antes de tudo posicionar manualmente a amostra para a posição inicial. Em seguida, passamos às configurações no painel do programa em LabView. Começamos pela esquerda no canto superior.

1 – O **deslocamento total** (em mícrons) deve ser ajustado com atenção, pois a amostra deverá permanecer o tempo todo dentro do campo gerado pelo eletroímã – valores usuais são entre 10000 e 14000 μm. O **passo**, que indica a distancia entre cada posição onde a amostra é medida, deve garantir uma boa resolução da curva – seu valor típico é 100 μm.

2 – Aqui ajustaremos **quantas vezes o campo da amostra será medido com esta na mesma posição**, para então obtermos uma média naquele local. Quanto mais alto o número de amostragens, menos interferência causará ruídos de média nula. Todavia, se a aquisição estiver sendo feita via GPIB, o tempo solicitado pode ser indesejavelmente alto.

3 – Este valor indica **quantas vezes a amostra fará a excursão dentro de um mesmo campo externo**. Seu principal objetivo é possibilitar, posteriormente, uma melhora do sinal através da média destas excursões. No entanto, mesmo para sinais intensos e com pouco ruído, o ideal é que tal valor seja no mínimo igual a 2; se imprevistos momentâneos comprometerem os valores de uma curva, haverá outra para substituí-la.

4 – A seguir, começamos a configurar os valores do campo externo B_ext. Aqui se encontra o controle de seus **valor máximo** e **passo**. Tomando como exemplo o que está na figura, teremos os valores 0, 200, 400, 600, ... , 2600, 2800, 3000 , 2800 , 2600 , etc. Devido a uma limitação do equipamento, o valor máximo não pode ultrapassar 3000 G.

5 – Uma curva de magnetização **completa**, conforme explicado inicialmente, geraria valores de campo externo positivos, negativos, e positivos novamente. Ou seja, no caso da figura: 0, 200, ... , 3000, 2800, ... , 0 , -200 , ... -3000 , -2800 , ... , 0 , 200 , ... 3000. Entretanto, há casos em que não se tem interesse em levantar a curva completa, seja porque se precisa avaliar a amostra apenas "por alto", seja por ganhar tempo para medir com passos mais precisos, etc. Por isso, foi inserida a opção de levantar-se a curva somente até o retorno a zero. Ou seja, a curva **parcial** geraria: 0, 200, ... , 2800, 3000, 2800, ... , 200, 0.

6 – Escolhe-se aqui onde e com que nome será **salvo o arquivo texto** com os valores de posição, campo da amostra na componente escolhida, e campo externo transversal.

7 – As opções da direita não são modificadas com freqüência, pois estão relacionadas geralmente com a disposição dos equipamentos, cujo ajuste é árduo. Este menu, por exemplo, indica de que **porta lógica** serão lidos os valores de campo (via conversor A/D ou via GPIB). Recomenda-se o conversor, pois este possui maior rapidez de resposta.

8 – Aqui se escolhe de qual **componente da ponteira do gaussímetro** serão lidos os valores de campo. Cabe ressaltar que sua convenção XYZ às vezes difere da utilizada pelas rotinas do MatLab.

9 – Quando se deseja medir uma amostra pela segunda vez, é interessante **desmagnetizá-la** primeiro, para que possíveis remanências magnéticas da medição anterior não interfiram na curva atual.

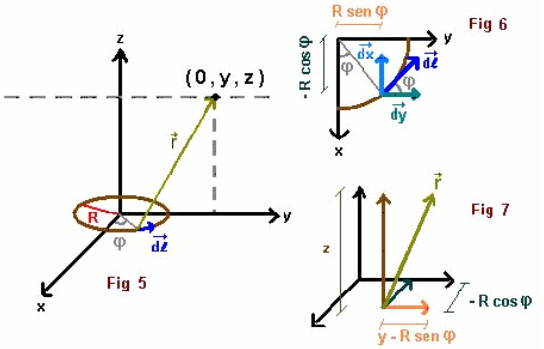
10 – Ajustado tudo, clica-se em **Levantar Curva** e a medição executará automaticamente até o fim.

Os indicadores no meio da tela mostrarão o que se passa em cada excursão, bem como a contagem de curvas e posições. Em caso de algum imprevisto, pode-se abortar o programa com o botão stop (o círculo vermelho no menu do próprio LabView, na parte superior da tela).

Apêndice 2: Simulação com Biot Savart

Conforme dito anteriormente, a amostra magnetizada pode ser modelada como um dipolo magnético centrado na origem. O motivo é simples: embora o material magnético não esteja presente somente nas bordas, a "corrente" de uma pequena área interna se cancela com a da adjacente. Apenas a corrente do trecho externo se preserva, conforme a figura mais abaixo.

A medição através dos magnetômetros se dá ao longo de várias posições duma reta paralela ao eixo y no plano yz. Tentaremos, pois, aplicar Biot-Savart para calcular a densidade de fluxo magnético no lugar geométrico em questão.



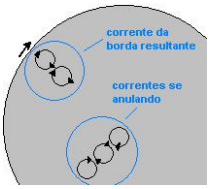


Fig 8: anulação das correntes internas, resultando no modelo da figura 5

Fig 6: $dl = R\,d\varphi \qquad dx = -dl\sin\varphi$

$d\bar{l} = dx\,\bar{i} + dy\,\bar{j} \qquad dy = dl\cos\varphi$

$d\bar{l} = -R\sin\varphi\,d\varphi\,\bar{i} + R\cos\varphi\,d\varphi\,\bar{j}$

Fig 7:

$$\bar{r} = -R\cos\varphi\,\bar{i} + (y - R\sin\varphi)\,\bar{j} + z\,\bar{k}$$

$$r = [R^2\cos^2\varphi + (y^2 - 2R\,y\sin\varphi + R^2\sin^2\varphi) + z^2]^{1/2} \Rightarrow r = (R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{1/2}$$

Biot-Savart:

$$dB_z = \frac{\mu_0 I}{4\pi}\frac{\left|(d\bar{l}\times\bar{r})_z\right|}{r^3} = \frac{\mu_0 I}{4\pi}\frac{(-R\sin\varphi\,d\varphi)(y - R\sin\varphi) + R\cos\varphi\,d\varphi\,R\cos\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}$$

$$dB_z = \frac{\mu_0 I}{4\pi}\frac{(-R\,y\sin\varphi\,d\varphi + R^2\sin^2 d\varphi + R^2\cos^2 d\varphi)}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}} = \frac{\mu_0 I}{4\pi}\frac{(-R\,y\sin\varphi + R^2)\,d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}$$

$$B_z = \int_0^{2\pi}\frac{\mu_0 I}{4\pi}\frac{(-R\,y\sin\varphi + R^2)\,d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}$$

$$B_z = \frac{\mu_0 I}{4\pi}\left[R^2\int_0^{2\pi}\frac{d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}} - R\,y\int_0^{2\pi}\frac{\sin\varphi\,d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}\right]$$

$$dB_y = \frac{\mu_0 I}{4\pi}\frac{\left|(d\bar{l}\times\bar{r})_y\right|}{r^3} = \frac{\mu_0 I}{4\pi}\frac{z\,R\sin\varphi\,d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}$$

$$B_y = \frac{\mu_0 I}{4\pi}z\,R\int_0^{2\pi}\frac{\sin\varphi\,d\varphi}{(R^2 + y^2 + z^2 - 2R\,y\sin\varphi)^{3/2}}$$

$$dB_x = \frac{\mu_0 I}{4\pi}\frac{\left|(d\bar{l}\times\bar{r})_x\right|}{r^3} = \frac{\mu_0 I}{4\pi}\frac{z\,R\cos\varphi\,d\varphi}{(R^2+y^2+z^2-2R\,y\sin\varphi)^{3/2}}$$

$$B_x = \frac{\mu_0 I}{4\pi} z\,R\int_0^{2\pi}\frac{\cos\varphi\,d\varphi}{(R^2+y^2+z^2-2R\,y\sin\varphi)^{3/2}}$$

Vale mencionar que esta última componente (X) resulta em zero, devido à simetria.

Conhecidos R, y, z, e I, "basta resolver" a integral definida para obtermos a expressão do campo ao longo da reta anteriormente desenhada.

Essa demonstração já constava no relatório da iniciação científica do ano anterior. Ali havia sido dito que tais integrais seriam resolvidas numericamente, e que para muito pontos seu processamento em MatLab poderia tornar-se lento. Felizmente, foi descoberto na literatura de eletromagnetismo (ver bibliografia no apêndice 5) que tais expressões poderiam ser simplificadas em integrais numéricas conhecidas e de cálculo mais rápido.

$$B_z = \frac{\mu_0 I}{2\pi}\frac{1}{\sqrt{(R+y)^2+z^2}}\left[K(k)+\frac{R^2-y^2-z^2}{(R-y)^2+z^2}E(k)\right]$$

$$B_y = \frac{\mu_0 I}{2\pi y}\frac{z}{\sqrt{(R+y)^2+z^2}}\left[-K(k)+\frac{R^2+y^2+z^2}{(R-y)^2+z^2}E(k)\right]$$

onde:

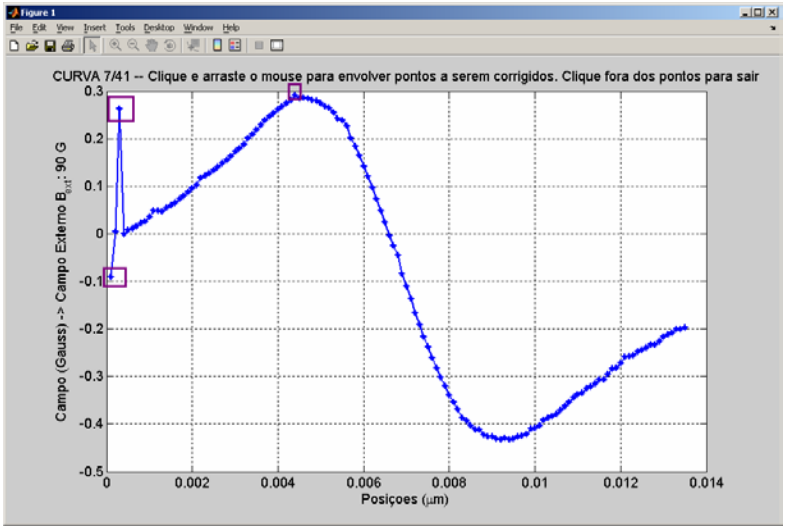
$$k^2 = \frac{4Ry}{(R+y)^2+z^2}$$

$$K(k) = \int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}$$

$$E(k) = \int_0^{\pi/2}\sqrt{1-k^2\sin^2\theta}\,d\theta$$

Apêndice 3: **Processamento no MatLab**

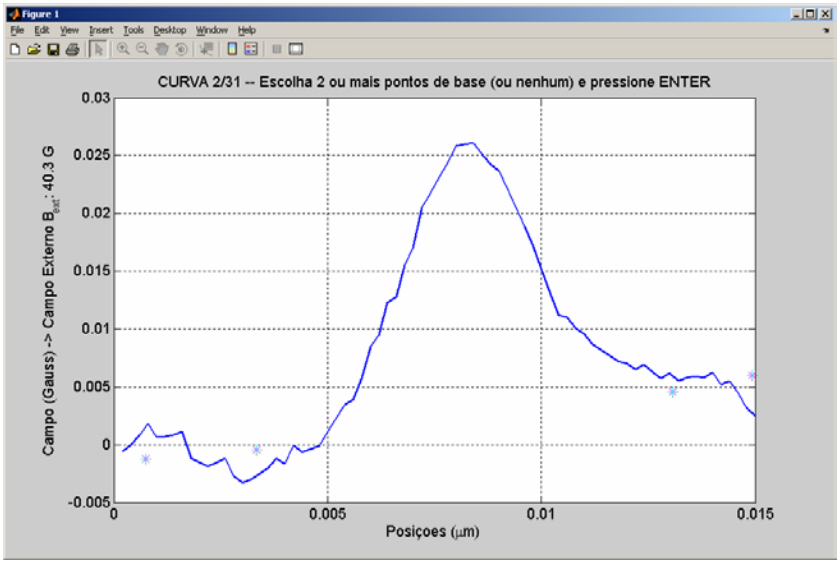
Quando finalizadas as medições, iniciamos o processo de melhora e comparação dos sinais. A primeira tarefa é eliminar as "impurezas" que podem prejudicar o sinal. Uma informação sempre importante é o valor do(s) pico(s), a qual é usada na centralização e na normalização da curva. Veja a figura abaixo.



Vê-se uma clara semelhança com a Figura 4, porém há um deslocamento no eixo X (a origem deveria estar pela parte central da curva) e no eixo Y (o máximo está em 0.3 G e o mínimo em -0.4 G). Ora, conhecendo a simetria garantida pela teoria, basta tomar a origem no ponto médio entre o máximo e o mínimo. Nenhum problema se não fossem perturbações como as presentes à esquerda. Neste caso não é grave, pois não se ultrapassou o máximo esperado mais à direita, mas ainda assim pode ser um desconforto para a comparação com dados teóricos.

Tentativas de correção automática via matemática computacional não mostraram resultados interessantes para uma variedade de distorções. A melhor solução encontrada foi a correção manual (no "olhômetro"), através da seleção dos picos indesejáveis. Conforme solicitado pelo titulo do gráfico, envolvemo-los e eles são imediatamente substituídos por pontos alinhados com os vizinhos que estão em posições corretas – uma aproximação linear válida quando houver várias medições por excursão da amostra.

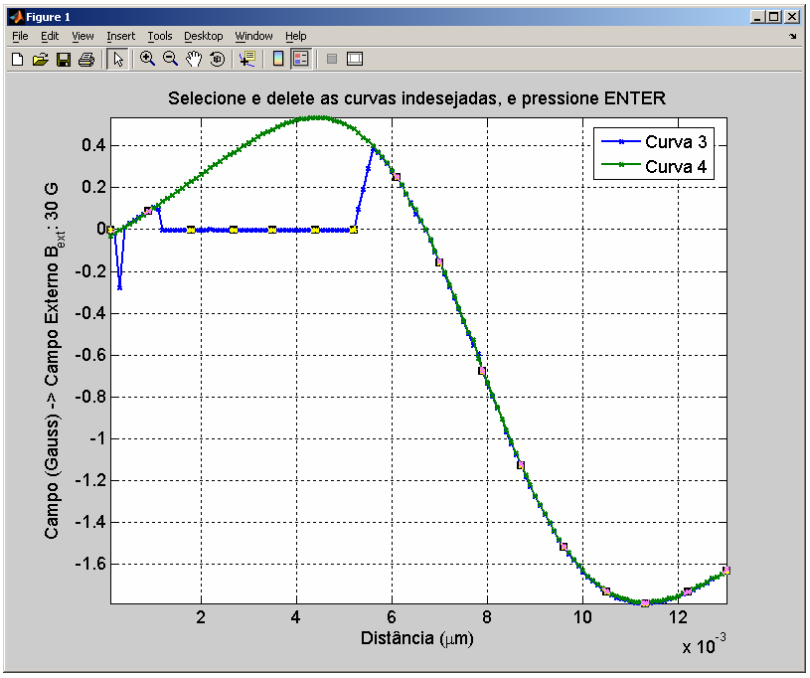
Enquanto essa primeira correção cuida de interferências de alta freqüência, uma segunda foi desenvolvida para as de baixa freqüência. Eventualmente, as curvas são acrescidas de um sinal que aumenta/diminui lentamente com o tempo, deixando sua base inclinada. Veja a figura a seguir.



Temos aqui então uma medição de outra componente do campo, a Z, paralela ao campo externo. Note que o sinal não retorna para o nível zero de onde veio. Mais uma vez, o conserto automático não oferece a melhora desejada para todos os casos. Por isso, solicita-se ao usuário a seleção de pontos da reta base, para que a curva "desça".

Esta parte é fundamental para as componentes Z, mesmo não havendo inclinações; o formato da curva não oferece a facilidade da componente Y em se encontrar a origem pela media entre os dois picos.

Por fim, mesmo que a curva não possa ser consertada, ela não atrapalhará o cálculo. Após os dois ajustes, todas as curvas de excursões sob um mesmo campo externo são reunidas para serem selecionadas. Ou melhor, para serem excluídas. Confira o exemplo.



Claramente, a curva azul sofreu com um problema momentâneo no medidor, e deverá ser eliminada. Clicamos nela e em DELETE no teclado.

As curvas restantes entram numa média, a qual é, enfim, comparada com o modelo teórico. Agora o objetivo é encontrar da amostra a corrente i equivalente de um dipolo, para então obtermos o momento magnético $m = i * A = i * \pi * R^2$ (lembrando que a amostra é circular). De acordo com as expressões encontradas de B_z e B_y, precisamos conhecer o valor de R (raio da amostra), z (distância transversal ao sensor), y (distância longitudinal ao sensor) para então obter i comparando-se os valores teórico e experimental de B.

R é facilmente obtido medindo-se o diâmetro da amostra assim que ela é preparada (e, claro, dividindo por 2 em seguida). Os valores de y são as posições da excursão, já corrigidas na centralização da curva. Resta então obter o z (chamado nas rotinas do MatLab de d, de distância). Tal parâmetro não é facilmente medido com precisão, pois a distância deve ser bem estreita para garantir um sinal mais intenso. Ou então, pode haver ainda um intervalo a mais entre a superfície da ponteira e seu sensor interno.

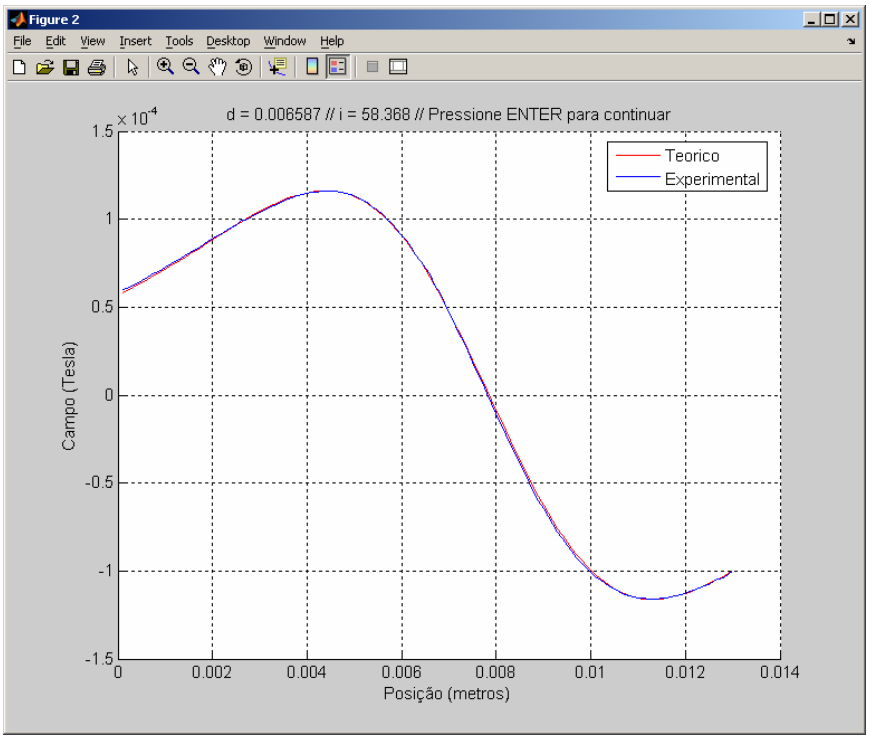
Ao invés de se fazer estimativas para d, foi usado o seguinte método:

1 – A corrente nada mais é que um fator multiplicado nas expressões de B. Ou seja, entre nossas incógnitas d e i, é a primeira que será responsável pelo formato da curva.

2 – Podemos inicialmente, assim sendo, normalizar tanto as curvas teóricas quanto as experimentais, de modo que seus picos coincidam sempre com 1. Isso pode ser feito para um valor qualquer de corrente.

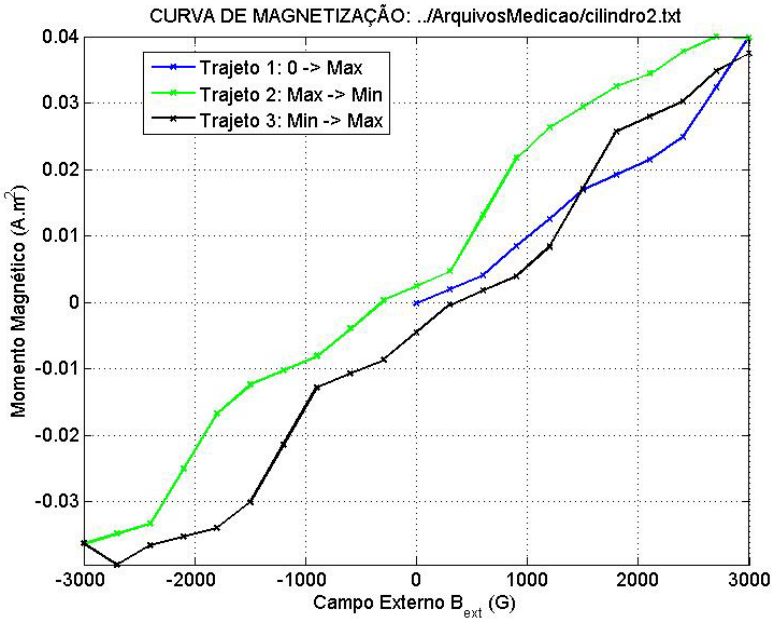
3 – Sem o efeito multiplicativo de i (devido à normalização), podemos encontrar d numa otimização numérica que, comparando com a curva experimental, tenta obter qual distância resulta no campo teórico mais próximo. Aqui se tentou usar algoritmos inteligentes (via derivadas, por exemplo), mas aparentemente nenhum deles convergiu bem. A solução mais eficiente foi mesmo a de força bruta: obtém-se valores teóricos para uma quantidade grande de distâncias (dentro de um intervalo estimado) e verifica-se qual dentre todas possui menor erro com o experimental. Com a velocidade de processamento atual dos computadores, o tempo gasto é razoavelmente curto.

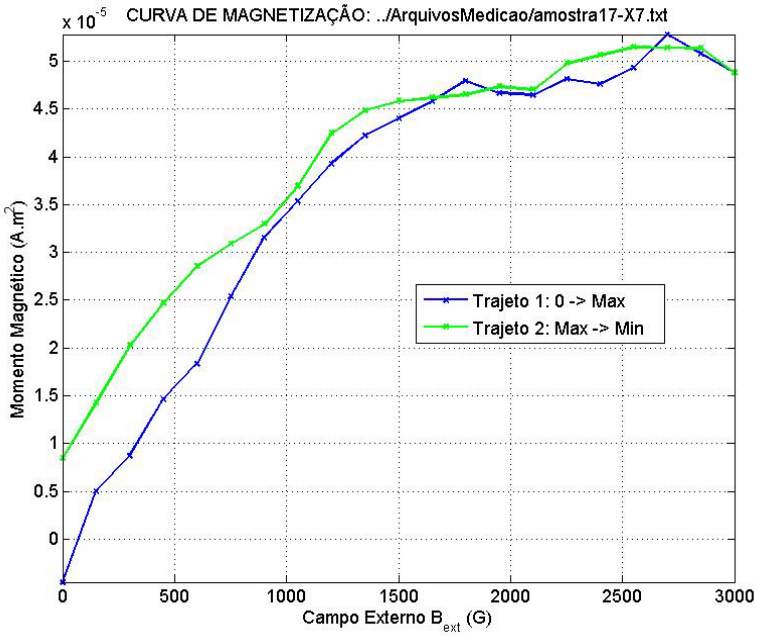
4 – Otimizado o parâmetro d, basta comparar o valor de pico de sua curva com a experimental, agora sem normalizações. O valor de i nos permite encontrar, finalmente, o momento de dipolo magnético. Veja o resultado final de uma otimização abaixo:



O processo é então repetido para todos os conjuntos de excursão de mesmo campo externo. Após processar as dezenas de gráficos, obtemos finalmente a curva de magnetização.

Abaixo seguem a curva completa de um cilindro magnético (usado para testar o sistema) e a curva parcial de uma amostra das partículas nanomagnéticas MagPrep. Como o cilindro possui muito mais material magnético que a amostra, os momentos magnéticos daquele foram algumas ordens de grandeza acima das deste. Além disso, nota-se também que a saturação não ocorre para campos externos por volta de 3000G no primeiro caso. Já no segundo, a saturação é nítida.





Apêndice 4: Rotinas (scripts) e funções em MatLab

```
%%%%%%%%%%%%%%%%%%%%% ROTINA PlotarCurvaMagnetizacao %%%%%%%%%%%%%%%%%%%%%%%
%
% Autor: Jan Krueger Siqueira , mar/2007
%
% Descrição da Rotina:
%   Levantar a Curva de Magnetização (m X B_ext - momento magnético por
%     densidade de fluxo magnético externo) de uma amostra magnética
%     através dos dados obtidos e salvos num arquivo, via LabView, relativo
%     a diversas excursões unidimensionais passando por um gaussímetro.
%   A cada excursão, a amostra passou pelas mesmas posiçoes, com um
%     determinado campo externo agindo sobre ela. Para cada uma das posições,
%     um valor de campo foi medido (descontando do campo externo). É
%     possível que cada campo externo tenha se repetido por N excursões, a
%     fim de se obterem valores médios de campo medidos, diminuindo o
```

```
%     efeito do ruído.
%   A curva B X Posição de cada excursão é exibido então na tela,
%     sequecialmente, e serão solicitadas correções manuais por parte do
%     usuário. É importante que, no final, o pico (no caso de componentes
%     Z tranversais) ou os 2 picos (no caso de componentes X ou Y
%     longitudinais) estejam nos locais devidos, pois serão utilizados na
%     centralização da curva.
%   As N curvas relativas ao mesmo campo externo são assim plotadas juntas,
%     com as correções aplicadas. Neste momento, pode-se excluir as que
%     ainda assim não estiverem boas. Pressionando ENTER, a média entre as
%     restantes é calculada e o valor da magnetização é obtido com base num
%     modelo teórico. O processo continua então, e se repete com as demais
%     curvas.
%   No fim, todos os pontos m X B_ext são plotados, e temos o gráfico de
%     magnetização.
%
% Arquivo de Entrada:
%   Espera-se que o arquivo .txt contenha 3 colunas numéricas: a primeira
%     representa as POSIÇÕES da amostra, a segunda os CAMPOS da amostra
%     naquele ponto e a terceira os CAMPOS EXTERNOS.
%   O exemplo de um arquivo gerado com essas características segue abaixo:
%
%                    (...)
%    0.000000      0.000000   274.000000   -
%    100.000000           0.000000   274.000000    |
%    200.000000          -0.015000   274.000000    | -- 1a. excursão com B_ext
%                    (...)                    |    de 274 G
%    12900.000000     0.008000       274.000000    |
%    13000.000000     0.008000       274.000000   -
%
%    0.000000      0.005000   274.000000   -
%    100.000000           0.009000   274.000000    |
%    200.000000           0.016000   274.000000    | -- 2a. excursão com B_ext
%                    (...)                    |    de 274 G
%    12900.000000     0.039000       274.000000    |
%    13000.000000     0.040000       274.000000   -
%
%    0.000000      0.000000   555.000000   -
%    100.000000           0.000000   555.000000    |
%    200.000000           0.001000   555.000000    | -- 1a. excursão com B_ext
%                    (...)                    |    de 555 G
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% limpa tela e variáveis, e fecha gráficos
    clc ;
    clear ;
    close all ;

% constantes
    DEBUG = 0 ;
    COLUNA_POSICOES = 1 ;
    COLUNA_CAMPO_AMOSTRA = 2 ;
    COLUNA_CAMPO_EXTERNO_Z = 3 ;

% constantes da medição da amostra
    NOME_DO_ARQUIVO = '../ArquivosMedicao/amostra17_X7.txt' ;
    COMPONENTE_MEDIDA_AMOSTRA = 'Y' ;
    if DEBUG
        NOME_DO_ARQUIVO = 'testeHisterese2.txt' ;
    end
    GANHO = 10 ;
    RAIO = 0.0015 ; % em metros

% aquisição dos dados do arquivo
    matriz_dadosArquivo = load( NOME_DO_ARQUIVO ) ;

    [ M , N ] = size( matriz_dadosArquivo ) ;
    if N ~= 3
        clear ;
        error( '*** ERRO *** Formatação de dados inválida no arquivo!' ) ;
    end

% ajuste dos valores de campo segundo o ganho usado
    matriz_dadosArquivo( : , COLUNA_CAMPO_AMOSTRA ) = ...
        matriz_dadosArquivo( : , COLUNA_CAMPO_AMOSTRA ) / GANHO ;
```

```
% posições que se repetem a cada excursão
    vet_indexPosicoesInicio = find( matriz_dadosArquivo( : , COLUNA_POSICOES ) == 0 ) ;
    totalPosicoes = vet_indexPosicoesInicio( 2 ) - vet_indexPosicoesInicio( 1 ) ;
    vet_posicoes = matriz_dadosArquivo( 2 : totalPosicoes , COLUNA_POSICOES ) ;
    vet_posicoes = vet_posicoes / 1e6 ; % de mícrons para metros

% obtenção dos campos externos de cada excursão
    vet_camposExternos = matriz_dadosArquivo( vet_indexPosicoesInicio , ...
                                              COLUNA_CAMPO_EXTERNO_Z ) ;

    primeiroCampoExterno = vet_camposExternos( 1 ) ;
    vet_indexOutrosCamposExternos = find( vet_camposExternos ~= ...
                                          primeiroCampoExterno ) ;
    qtdExcursoesPorCampoExterno = vet_indexOutrosCamposExternos( 1 ) - 1 ;
       % isto é, se o primeiro campo externo não se repete na 4ª excursão,
       %   tivemos 3 excursões por campo externo

% preparação para plotagem das curvas
    numCurvaAtual = 1 ; % apenas uma contagem para o usuário saber quantas curvas ele tem que
corrigir
    qtdCurvas = length( vet_indexPosicoesInicio ) ;
    numExcursao = 1 ;   % conta excursões de um mesmo campo externo B_ext

    figure ;
    set( gca , 'fontsize' , 13 ) ;
    set( gcf , 'Position' , [200 200 1000 600] ) ;
    matriz_camposMesmoB_ext = zeros( totalPosicoes - 1 , qtdExcursoesPorCampoExterno ) ;
    matriz_legenda = [ ] ;
    vet_momentosDipolo = [ ] ;

% plotar todas as excursões, uma a uma, para corrigirmos os desníveis
    for i = vet_indexPosicoesInicio'
        vet_trechoExcursao = ( i + 1 ) : ( i + totalPosicoes - 1 ) ;

        vet_camposExcursaoAtual = matriz_dadosArquivo( vet_trechoExcursao , ...
                                                       COLUNA_CAMPO_AMOSTRA ) ;

    % primeira correção: picos indesejáveis serão manualmente selecionados e eliminados
        plot( vet_posicoes ,  vet_camposExcursaoAtual , '-*b' ,'LineWidth' , 2 ) ;
        set( gcf , 'Position' , [200 200 1000 600] ) ;
        set( gca , 'fontsize' , 13 ) ;
        grid on ;
        xlabel( 'Posiçoes (\mum)' ) ;
        ylabel( [ 'Campo (Gauss) -> Campo Externo B_e_x_t: ' ...
                  num2str( vet_camposExternos( numCurvaAtual ) ) ' G' ] ) ;
        string_nomeArquivo = strrep( NOME_DO_ARQUIVO , '_' , '-' ) ;
        title( [ 'CURVA ' num2str( numCurvaAtual ) '/' num2str( qtdCurvas ) ...
                 ' -- Clique e arraste o mouse para envolver pontos a serem corrigidos.' ...
                 ' Clique fora dos pontos para sair' ] ) ;

        vet_retangulo = getrect( ) ;
        vet_indicesPontosDentroRetangulo = IndicesPontosDentroRetangulo( vet_posicoes , ...
                                           vet_camposExcursaoAtual , vet_retangulo ) ;

              while ~isempty( vet_indicesPontosDentroRetangulo )
            indiceFim = length( vet_indicesPontosDentroRetangulo ) ;
            if vet_indicesPontosDentroRetangulo( 1 ) == 1
                x1 = vet_posicoes( vet_indicesPontosDentroRetangulo( indiceFim ) + 2 ) ;
                y1 = vet_camposExcursaoAtual( vet_indicesPontosDentroRetangulo( indiceFim ) +
2 ) ;
            else
                x1 = vet_posicoes( vet_indicesPontosDentroRetangulo( 1 ) - 1 ) ;
                y1 = vet_camposExcursaoAtual( vet_indicesPontosDentroRetangulo( 1 ) - 1 ) ;
            end
            indiceFim = length( vet_indicesPontosDentroRetangulo ) ;
            if vet_indicesPontosDentroRetangulo( indiceFim ) == length( vet_posicoes )
                x2 =  vet_posicoes( vet_indicesPontosDentroRetangulo( 1 ) - 2 ) ;
                y2 =  vet_camposExcursaoAtual( vet_indicesPontosDentroRetangulo( 1 ) - 2 ) ;
            else
                x2 =  vet_posicoes( vet_indicesPontosDentroRetangulo( indiceFim ) + 1 ) ;
                y2 =  vet_camposExcursaoAtual( vet_indicesPontosDentroRetangulo( indiceFim ) +
1 ) ;
            end

            % aproximaçao linear: pontos englobados são alinhados com os
            %   vizinhos de fora: o da esquerda e o da direita
                [ vet_coefReta , lixo ] = polyfit( [ x1 x2 ] , [ y1 y2 ] , 1 ) ;
```

```
                vet_camposExcursaoAtual( vet_indicesPontosDentroRetangulo ) = ...
                       polyval( vet_coefReta , vet_posicoes( vet_indicesPontosDentroRetangulo
) ) ;

            plot( vet_posicoes ,  vet_camposExcursaoAtual , '-*b' ,'LineWidth' , 2 ) ; % curva
corrigida
            set( gcf , 'Position' , [200 200 1000 600] ) ;
            set( gca , 'fontsize' , 13 ) ;
            grid on ;
            xlabel( 'Posiçoes (\mum)' ) ;
            ylabel( [ 'Campo (Gauss) -> Campo Externo B_e_x_t: ' ...
                  num2str( vet_camposExternos( numCurvaAtual ) ) ' G' ] ) ;
            title( [ 'CURVA ' num2str( numCurvaAtual ) '/' num2str( qtdCurvas ) ...
                     ' -- Clique e arraste o mouse para envolver pontos a serem corrigidos.'
...
                     ' Clique fora dos pontos para sair' ] ) ;

            vet_retangulo = getrect( ) ;
            vet_indicesPontosDentroRetangulo = IndicesPontosDentroRetangulo( vet_posicoes ,
...
                                                vet_camposExcursaoAtual , vet_retangulo ) ;
        end

    % segunda correção: curva poderá ser nivelada caso haja uma inclinação
    % indesejável
        plot( vet_posicoes ,  vet_camposExcursaoAtual , 'b' , 'LineWidth' , 2 ) ;
        set( gcf , 'Position' , [200 200 1000 600] ) ;
        grid on ;
        xlabel( 'Posiçoes (\mum)' ) ;
        ylabel( [ 'Campo (Gauss) -> Campo Externo B_e_x_t: ' ...
                  num2str( vet_camposExternos( numCurvaAtual ) ) ' G' ] ) ;
        title( [ 'CURVA ' num2str( numCurvaAtual ) '/' num2str( qtdCurvas ) ...
                 ' -- Escolha 2 ou mais pontos de base (ou nenhum) e pressione ENTER' ] ) ;

        [ x , y ] = getpts ;

    % conserta possivel rampa, calculando a reta entre os ponto selecionados
    %  e depois subtraindo gráfico dessa reta
        if length( x ) >= 2
            [ vet_coefReta , lixo ] = polyfit( x , y , 1 ) ;

            vet_camposExcursaoAtual = vet_camposExcursaoAtual - ...
                                 polyval( vet_coefReta , vet_posicoes ) ;
        end

        matriz_legenda = strvcat( matriz_legenda , ...
                                 [ 'Curva ' num2str( numCurvaAtual ) ] ) ;

        matriz_camposMesmoB_ext( : , numExcursao ) = vet_camposExcursaoAtual ;

    % plotar curvas de excursões juntas, de cada repetiçao no B_ext,
    %   para apagar as ruins e fazer a média
        if numExcursao == qtdExcursoesPorCampoExterno | ...
           numCurvaAtual == length( vet_indexPosicoesInicio )

            plot( vet_posicoes , matriz_camposMesmoB_ext , '-x' , 'LineWidth' , 2 ) ;

            set( gca , 'fontsize' , 13 ) ;
            xlabel( 'Distância (\mum)' ) ;
            NOME_DO_ARQUIVO = strrep( NOME_DO_ARQUIVO , '_' , '-' ) ;
            ylabel( [ 'Campo (Gauss) -> Campo Externo B_e_x_t: ' ...
                  num2str( vet_camposExternos( numCurvaAtual ) ) ' G' ] ) ;
            title( 'Selecione e delete as curvas indesejadas, e pressione ENTER' ) ;

            set( gcf , 'Position' , [200 200 800 600] ) ;
            axis tight ;
            grid on ;
            plotedit on ;

            legend( matriz_legenda ) ;

            pause ; % aguarda ENTER do usuário

            axis tight ;

            numExcursao = 0 ;
            matriz_legenda = [ ] ;
```

```
            matriz_camposDesejados = get( get( gca , 'children' ) , 'YData' ) ;
            qtdCurvasRestantes = length( matriz_camposDesejados ) ;

        % calculando média das curvas restantes
            if iscell( matriz_camposDesejados )
                matriz_camposDesejados = cell2mat( matriz_camposDesejados ) ;
                vet_campoMedio = mean( matriz_camposDesejados ) ;
            else
                vet_campoMedio = matriz_camposDesejados ;
            end

        % calculando o momento de dipolo teórico aproximado
            vet_campoMedio = vet_campoMedio / 1e4 ; % de gauss para tesla

            if length( vet_campoMedio ) == 0
                vet_momentosDipolo = [ vet_momentosDipolo NaN ] ;
            else
                momento = CalcularMomentoDipolo( vet_posicoes' , vet_campoMedio' , ...
                                                 RAIO , COMPONENTE_MEDIDA_AMOSTRA ) ;
                vet_momentosDipolo = [ vet_momentosDipolo momento ]
            end
        end

        numCurvaAtual = numCurvaAtual + 1 ;
        numExcursao = numExcursao + 1 ;
    end

% tirando repetições do campo externo devido às múltiplas excursões
    vet_camposExternos = vet_camposExternos( 1 : qtdExcursoesPorCampoExterno : ...
                                        length( vet_camposExternos ) ) ;

% tirando pontos NaN dos 2 vetores
    vet_indexNan = find( isnan( vet_momentosDipolo ) ) ;
    vet_camposExternos( vet_indexNan ) = [ ] ;
    vet_momentosDipolo( vet_indexNan ) = [ ] ;

% dividindo os vetores em 3 trajetos
    vet_derivadasCamposExternos = diff( vet_camposExternos ) ;
    vet_indexDerivadasNegativas = find( vet_derivadasCamposExternos < 0 ) ;

    if numel( vet_indexDerivadasNegativas ) ~= 0
        indexRetrocessoCampo = vet_indexDerivadasNegativas( 1 ) ;
        indexUltimoNegativo = vet_indexDerivadasNegativas( ...
                              length( vet_indexDerivadasNegativas ) ) ;
        indexRetornoAumentoCampo = indexUltimoNegativo + 1 ;
        indexFim = length( vet_camposExternos ) ;
    else
        indexRetrocessoCampo = length( vet_camposExternos ) ;
        indexUltimoNegativo = -1 ;
        indexRetornoAumentoCampo = -2 ;
        indexFim = -3 ;
    end

    vet_indexTrajeto1 = 1 : indexRetrocessoCampo ;
    vet_indexTrajeto2 = indexRetrocessoCampo : indexRetornoAumentoCampo ;
    vet_indexTrajeto3 = indexRetornoAumentoCampo : indexFim ;

% levantando enfim a curva de magnetização
    close ;
    figure ;
    plot( vet_camposExternos( vet_indexTrajeto1 )' , ...
          vet_momentosDipolo( vet_indexTrajeto1 ) , '-bx' , 'LineWidth' , 2 ) ;
    hold ;
    plot( vet_camposExternos( vet_indexTrajeto2 )' , ...
          vet_momentosDipolo( vet_indexTrajeto2 ) , '-gx' , 'LineWidth' , 2 ) ;
    plot( vet_camposExternos( vet_indexTrajeto3 )' , ...
          vet_momentosDipolo( vet_indexTrajeto3 ) , '-kx' , 'LineWidth' , 2 ) ;

    set( gca , 'fontsize' , 13 ) ;
    xlabel( 'Campo Externo B_e_x_t (G)' ) ;
    ylabel( 'Momento Magnético (A.m^2)' ) ;
    string_nomeArquivo = strrep( NOME_DO_ARQUIVO , '_' , '-' ) ;
    titulo = [ 'CURVA DE MAGNETIZAÇÃO: ' string_nomeArquivo  ] ;
    title( titulo ) ;
    legend( 'Trajeto 1: 0 -> Max' , 'Trajeto 2: Max -> Min' , ...
            'Trajeto 3: Min -> Max' ) ;
```

```
    set( gcf , 'Position' , [200 200 800 600] ) ;
    axis tight ;
    grid on ;

    hold ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA ROTINA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vet_indices = IndicesPontosDentroRetangulo ( vet_coordenadasX , vet_coordenadasY ,
vet_retangulo )
    xMinimo = vet_retangulo( 1 ) ;
    yMinimo = vet_retangulo( 2 ) ;
    largura = vet_retangulo( 3 ) ;
    altura = vet_retangulo( 4 ) ;
    xMaximo = xMinimo + largura ;
    yMaximo = yMinimo + altura ;

    vet_indices = find( vet_coordenadasX > xMinimo & vet_coordenadasX < xMaximo & ...
                        vet_coordenadasY > yMinimo & vet_coordenadasY < yMaximo ) ;

%%%%%%%%%%%%%%%%%%%%%%%% FUNÇÃO CalcularMomentoDipolo %%%%%%%%%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , jul/2007
%
%  Descrição da Função:
%    Calcula o momento de dipolo magnético com base nos dados experimentais
%      de densidade de fluxo magnético. Tais valores são comparados com os
%      teóricos, buscando obter o modelo de dipolo com corrente i mais
%      próximo.
%    Caso a componente de densidade de fluxo seja a Y, será feita uma
%      centralização vertical, para manter os picos simétricos. Caso seja a
%      Z, espera-se que a curva já tenha sido previamente centralizada
%      nesse aspecto. Em ambos os casos, ocorre também a centralização
%      vertical.
%
%  Chamada da Função:
%       momentoDipolo = CalcularMomentoDipolo ( vet_dadosX , vet_dadosY , raio
%                                             , str_componenteMedida )
%
%  Parâmetros Recebidos:
%    vet_dadosX             - posições na excursão             -> em metros
%    vet_dadosY             - densidades de fluxo na excursão  -> em Tesla
%    raio                   - raio da amostra                  -> em metros
%    str_componenteMedida   - 'Y' ou 'Z'
%
%  Valor Retornado:
%    momentoDipolo          - valor do componente Y de B       -> em Ampère
%                                                                 * metro^2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function momentoDipolo = CalcularMomentoDipolo ( vet_dadosX , vet_dadosY , raio ,
str_componenteMedida )

DEBUG = 1 ;

% desliga temporariamente warnings, por causa de divisões por zero
    warning off all ;

% consertando a curva
    if strcmp( str_componenteMedida , 'Y' )
        funcaoASerOtimizada = @CalcularCampoYNormalizadoDipolo ;

        [ maximoY , indiceMaxCurva ] = max( vet_dadosY ) ;
        [ minimoY , indiceMinCurva ] = min( vet_dadosY ) ;
        if indiceMinCurva < indiceMaxCurva
            sinalCurva = 1 ;
        else
            sinalCurva = -1 ;
        end

        vet_dadosXCorrigido = vet_dadosX - ( vet_dadosX( indiceMaxCurva ) + ...
                                             vet_dadosX( indiceMinCurva ) ) / 2 ;

    % normaliza campo para otimização da distância, a qual influencia na forma
    %   da curva
        vet_dadosYCentralizado = vet_dadosY - ( maximoY + minimoY ) / 2 ;
```

```
        vet_dadosYNormalizado = vet_dadosYCentralizado / max( vet_dadosYCentralizado ) ;
    elseif strcmp( str_componenteMedida , 'Z' )
        funcaoASerOtimizada = @CalcularCampoZNormalizadoDipolo ;

        [ moduloPico indicePico ] = max( abs( vet_dadosY ) ) ;
        sinalPico = vet_dadosY( indicePico ) / moduloPico ;
        sinalCurva = sinalPico ;

        vet_dadosXCorrigido = vet_dadosX - vet_dadosX( indicePico ) ;

    % normaliza campo para otimização da distância, a qual influencia na forma
    %   da curva
    % espera-se aqui que o Y já tenha sido centralizado previamente
        vet_dadosYCentralizado = vet_dadosY ;
        vet_dadosYNormalizado = vet_dadosYCentralizado / moduloPico ;
    else
        error( '*** ERRO *** Parametro 4 (str_componenteMedida) deve ser Y ou Z' ) ;
    end

% otimizaçao da distância
    correnteUnitaria = 1 ;
    valorInferior = 0.0001 ;
    valorSuperior = 0.05 ;

    dOtimizado = MinimizarErro( funcaoASerOtimizada , valorInferior , ...
                                valorSuperior , vet_dadosYNormalizado , ...
                                vet_dadosXCorrigido , correnteUnitaria , raio , sinalCurva )
;

% a corrente é apenas um fator multiplicativo
    if strcmp( str_componenteMedida , 'Y' )
        vet_yTeorico = CalcularCampoYDipolo( dOtimizado , vet_dadosXCorrigido ,
correnteUnitaria , raio , sinalCurva ) ;
    else
        vet_yTeorico = CalcularCampoZDipolo( dOtimizado , vet_dadosXCorrigido ,
correnteUnitaria , raio , sinalCurva ) ;
    end

    iDipolo = max( abs( vet_dadosYCentralizado ) ) / max( abs( vet_yTeorico ) ) ;

if DEBUG
    vet_yTeorico2 = vet_yTeorico * iDipolo ;
    figure ;
    set( gca , 'fontsize' , 11 ) ;
    set( gcf , 'Position' , [200 200 800 600] ) ;
    hold ;
    plot( vet_dadosX , vet_yTeorico2 , 'r' ) ;
    plot( vet_dadosX , vet_dadosYCentralizado , 'b' ) ;
    hold ;
    title( [ 'd = ' num2str( dOtimizado ) ' // i = ' num2str( iDipolo ) ...
             ' // Pressione ENTER para continuar' ] ) ;
    legend( 'Teorico' , 'Experimental' ) ;
    xlabel( 'Posição (metros)' ) ;
    ylabel( 'Campo (Tesla)' ) ;
    grid ;
    pause ;
    close ;
end

if strcmp( str_componenteMedida , 'Y' )
    sinalCurva = -sinalCurva ;
end

% com a corrente e o raio, encontra-se então m = i * S = i * pi * raio²
    momentoDipolo = iDipolo * pi * raio ^ 2 * sinalCurva ;

% reabilita warnings
    warning on all ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% FUNÇÃO MinimizarErro %%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , jul/2007
%
%  Descrição da Funçao:
%    Parecida com a funçao lsqcurvefit.
```

```
%    Dada uma função Y = f(X), tenta encontrar um valor "X" que resulte no
%      (ou seja próximo do) valor "Y" desejado.
%    "X" e "Y" podem ser vetores.
%    O erro do "X" encontrado varia bastante dependendo da funçao.
%
%      Chamada da Função:
%        X = MinimizarErro2( funcao , valorInf , valorSup , respEsp , P1 ,
%                          P2 , ... ) ;
%
%  Parâmetros Recebidos:
%    funcao   - função a ser "otimizada"
%    valorInf - limite inferior de onde se deseja encontrar a resposta
%    valorSup - limite superior de onde se deseja encontrar a resposta
%    respEsp  - resposta ("Y") que se espera encontrar em "funçao"
%               com o parâmetro procurado ("X")
%    P1,P2,...- (OPCIONAL) parâmetros extras a serem passados para "função"
%
%    A funçao passada será chamada: funcao( X , P1 , P2 , ... )
%                                ou: funcao( X )
%
%  Valor Retornado:
%    X        - parâmetro procurado
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X = MinimizarErro ( funcao , valorInferior , valorSuperior , ...
                             respostaEsperada , varargin )

% constantes
    QTD_TENTATIVAS = 2000 ;

% checar parâmetros
    if nargin < 4
        error( '*** ERRO *** MinizarErro3 requer 4 ou mais parâmetros!' ) ;
    end

    if valorInferior >= valorSuperior
        error( '*** ERRO *** Limite inferior >= Limite Superior!!!' ) ;
    end

% dados iniciais
    valoresTeste1 = valorInferior : ...
                ( ( valorSuperior - valorInferior ) / QTD_TENTATIVAS ) : valorSuperior ;

    erroAnterior = inf ;
    indiceMenorErro = 0 ;

% chuta (arbitra) QTD_TENTATIVAS valores na faixa especificada e encontra o de menor erro
    for i = 1 : length( valoresTeste1 )

        respostaObtida = funcao( valoresTeste1( i ) , varargin{ : } ) ;
        erroObtido = sum( abs( respostaObtida' - respostaEsperada ) ) ;

        if erroObtido < erroAnterior
            erroAnterior = erroObtido ;
            indiceMenorErro = i ;
            aux = respostaObtida ;
        end
    end

    X = valoresTeste1( indiceMenorErro ) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%% FUNÇÃO CalcularCampoYNormalizadoDipolo %%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , abr/2007
%
%  Descrição da Função:
%    Calcula a coordenada Y (longitudinal) da densidade de fluxo magnético
%      B dum dipólo, num ponto qualquer dado em coordenadas cilíndricas.
%    Como o sentido do campo depende da localização do ponto em relação ao
%      sentido da corrente, valores negativos de rho são aceitos. Além
%      disso, um quinto parâmetro opcional pode ser usado para a
%      orientação do sinal.
%    Esta função normaliza (divide todos os valores pelo máximo) os campos,
%      pois ela é utilizada numa otimização onde apenas o formato da curva
```

```
%      importa.
%
%  Chamada da Função:
%       By = CalcularCampoYNormalizadoDipolo( z , rho , i , a ) ;
%
%  Parâmetros Recebidos:
%    z   - coordenada Z do ponto                          -> em metros
%    rho - coordenada RHO do ponto (distância ao eixo Z) -> em metros
%    i   - corrente que circula no dipolo
%          (sentido anti-horário)                         -> em Ampères
%    a   - raio da espira (dipolo)                        -> em metros
%
%    orientacao (opcional) - se omitido ou igual a 1, By assume valores
%                            negativos para rhos com sinal diferente de i;
%                            se igual a -1, ocorre o oposto
%
%  Valor Retornado:
%    By  - valor do componente normalizada Y de B        -> em Tesla
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function By = CalcularCampoYNormalizadoDipolo ( z , rho , i , a , orientacao )

    By = CalcularCampoYDipolo( z , rho , i , a , orientacao ) ;
    sizeBy = size( By ) ;
    numeroLinhasBy = sizeBy( 1 ) ;
    mat_maxBy = ones( numeroLinhasBy , 1 ) * max( By ) ;
    By = By ./ mat_maxBy ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%% FUNÇÃO CalcularCampoYDipolo %%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , abr/2007
%
%  Descrição da Função:
%    Calcula a coordenada Y da densidade de fluxo magnético B dum dipólo,
%      num ponto qualquer dado em coordenadas cilíndricas.
%    Como o sentido do campo depende da localização do ponto em relação ao
%      sentido da corrente, valores negativos de rho são aceitos. Além
%      disso, um quinto parâmetro opcional pode ser usado para a
%      orientação do sinal.
%
%  Chamada da Função:
%       By = CalcularCampoYNormalizadoDipolo( z , rho , i , a ) ;
%
%  Parâmetros Recebidos:
%    z   - coordenada Z do ponto                          -> em metros
%    rho - coordenada RHO do ponto (distância ao eixo Z) -> em metros
%    i   - corrente que circula no dipolo
%          (sentido anti-horário)                         -> em Ampères
%    a   - raio da espira (dipolo)                        -> em metros
%
%    orientacao (opcional) - se omitido ou igual a 1, By assume valores
%                            negativos para rhos com sinal diferente de i;
%                            se igual a -1, ocorre o oposto
%
%  Valor Retornado:
%    By  - valor do componente Y de B                     -> em Tesla
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function By = CalcularCampoYDipolo ( z , rho , i , a , orientacao )

    MI_ZERO = 4e-7 * pi ;

    if nargin ~= 4 & nargin ~= 5
        error( '*** ERRO *** Devem ser passados 4 ou 5 argumentos!' ) ;
    end

    dimensaoZ = size( z ) ;
    dimensaoRho = size( rho ) ;
    dimensaoI = size( i ) ;
    dimensaoA = size( a ) ;

    if ( dimensaoZ == dimensaoRho | dimensaoRho == 1 | dimensaoZ == 1 ) & ...
       ( dimensaoZ == dimensaoI | dimensaoI == 1 | dimensaoZ == 1 ) & ...
```

```
       ( dimensaoZ == dimensaoA | dimensaoA == 1 | dimensaoZ == 1 ) & ...
       ( dimensaoRho == dimensaoI | dimensaoI == 1 | dimensaoRho == 1 ) & ...
       ( dimensaoRho == dimensaoA | dimensaoA == 1 | dimensaoRho == 1 ) & ...
       ( dimensaoI == dimensaoA | dimensaoA == 1 | dimensaoI == 1 )

      % satisfeito tudo isso, nenhum problema
    else
        error( [ '*** ERRO *** Se dois dos quatro parametros não possuirem mesma dimensão, ' ,
...
                 'pelo menos um deles deverá ser escalar!' ] ) ;
    end

    if ~isreal( z )
        error( '*** ERRO *** Parametro 1 ("z") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( rho )
        error( '*** ERRO *** Parametro 2 ("rho") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( i )
        error( '*** ERRO *** Parametro 3 ("i") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( a )
        error( '*** ERRO *** Parametro 4 ("a") inválido! Não pode ser complexo!' ) ;
    end
    if a > 0
        % OK, todos os valores de a são positivos
    else
        error( '*** ERRO *** Parametro 4 ("a") inválido! Não pode ser negativo!' ) ;
    end

    if nargin == 4
        orientacao = 1 ;
    elseif orientacao == 1 | orientacao == -1
        % OK!
    else
        error( '*** ERRO *** Parametro 5 ("orientacao") inválido! Deve ser 1 ou -1!' ) ;
    end

% dá erro se rho for negativo (exemplo: usando coordenada y no lugar)
    sinal = rho ./ abs( rho ) ;
    rho = abs( rho ) ;

    k =  4 * a .* rho ./ ( ( a + rho ) .^ 2 + z .^ 2 ) ;
    [ K , E ] = ellipke( k ) ;

% componente Y
    termo1 = MI_ZERO * i .* z ./ ( 2 * pi * rho .* sqrt( ( a + rho ) .^ 2 + z .^ 2 ) ) ;
    termo2 = -K + ( a .^ 2 + rho .^ 2 + z .^ 2 ) .* E ./ ...
                 ( ( a - rho ) .^ 2 + z .^ 2 ) ;

% consertando sinal para rhos negativos
    By = termo1 .* termo2 .* sinal .* orientacao ;

% caso rho = 0 resulta numa divisao 0 / 0 = NaN, cujo limite é conhecido como 0
    By( find( isnan( By ) ) ) = 0 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%% FUNÇÃO CalcularCampoZNormalizadoDipolo %%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , jan/2007
%
%  Descrição da Função:
%    Calcula a coordenada Z (transversal) da densidade de fluxo magnético
%      B dum dipólo, num ponto qualquer dado em coordenadas cilíndricas.
%      Valores negativos de rho são aceitos, porém apenas o módulo
%      influencia no cálculo.
%    Esta função normaliza (divide todos os valores pelo máximo em módulo)
%      os campos, pois ela é utilizada numa otimização onde apenas o
%      formato da curva importa.
%
%  Chamada da Função:
%       Bz = CalcularCampoZNormalizadoDipolo( z , rho , i , a , orientacao ) ;
%
```

```
%  Parâmetros Recebidos:
%    z   - coordenada Z do ponto                          -> em metros
%    rho - coordenada RHO do ponto (distância ao eixo Z) -> em metros
%    i   - corrente que circula no dipolo
%          (sentido anti-horário)                         -> em Ampères
%    a   - raio da espira (dipolo)                        -> em metros
%
%    orientacao (opcional) - se omitido ou igual a 1, Bz tem seu pico como
%                            positivo;
%                            se igual a -1, o pico é negativo
%
%  Valor Retornado:
%    Bz   - valor do componente normalizada Z             -> em Tesla
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Bz = CalcularCampoZNormalizadoDipolo ( z , rho , i , a , orientacao )

    Bz = CalcularCampoZDipolo( z , rho , i , a , orientacao ) ;
    sizeBz = size( Bz ) ;
    numeroLinhasBz = sizeBz( 1 ) ;
    mat_maxBz = ones( numeroLinhasBz , 1 ) * max( abs( Bz ) ) ;
    Bz = Bz ./ mat_maxBz ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%% FUNÇÃO CalcularCampoZDipolo %%%%%%%%%%%%%%%%%%
%
%  Autor: Jan Krueger Siqueira , jan/2007
%
%  Descrição da Função:
%    Calcula a coordenada Z da densidade de fluxo magnético B dum dipólo,
%      num ponto qualquer dado em coordenadas cilíndricas. Valores
%      negativos de rho são aceitos, porém apenas o módulo influencia no
%      cálculo.
%
%  Chamada da Função:
%       Bz = CalcularCampoZDipolo( z , rho , i , a ) ;
%
%  Parâmetros Recebidos:
%    z   - coordenada Z do ponto                          -> em metros
%    rho - coordenada RHO do ponto (distância ao eixo Z) -> em metros
%    i   - corrente que circula no dipolo
%          (sentido anti-horário)                         -> em Ampères
%    a   - raio da espira (dipolo)                        -> em metros
%
%    orientacao (opcional) - se omitido ou igual a 1, Bz tem seu pico como
%                            positivo;
%                            se igual a -1, o pico é negativo
%
%  Valor Retornado:
%    Bz   - valor do componente Z do campo             -> em Ampère * metro
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Bz = CalcularCampoZDipolo ( z , rho , i , a , orientacao )

    MI_ZERO = 4e-7 * pi ;

    if nargin ~= 4 & nargin ~= 5
        error( '*** ERRO *** Devem ser passados 4 ou 5 argumentos!' ) ;
    end

    dimensaoZ = size( z ) ;
    dimensaoRho = size( rho ) ;
    dimensaoI = size( i ) ;
    dimensaoA = size( a ) ;

    if ( dimensaoZ == dimensaoRho | dimensaoRho == 1 | dimensaoZ == 1 ) & ...
       ( dimensaoZ == dimensaoI | dimensaoI == 1 | dimensaoZ == 1 ) & ...
       ( dimensaoZ == dimensaoA | dimensaoA == 1 | dimensaoZ == 1 ) & ...
       ( dimensaoRho == dimensaoI | dimensaoI == 1 | dimensaoRho == 1 ) & ...
       ( dimensaoRho == dimensaoA | dimensaoA == 1 | dimensaoRho == 1 ) & ...
       ( dimensaoI == dimensaoA | dimensaoA == 1 | dimensaoI == 1 )

       % satisfeito tudo isso, nenhum problema
    else
```

```
        error( [ '*** ERRO *** Se dois dos quatro parametros não possuirem mesma dimensão, ' ,
...
                 'pelo menos um deles deverá ser escalar!' ] ) ;
    end

    if ~isreal( z )
        error( '*** ERRO *** Parametro 1 ("z") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( rho )
        error( '*** ERRO *** Parametro 2 ("rho") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( i )
        error( '*** ERRO *** Parametro 3 ("i") inválido! Não pode ser complexo!' ) ;
    end

    if ~isreal( a )
        error( '*** ERRO *** Parametro 4 ("a") inválido! Não pode ser complexo!' ) ;
    end
    if a > 0
        % OK, todos os valores de a são positivos
    else
        error( '*** ERRO *** Parametro 4 ("a") inválido! Não pode ser negativo!' ) ;
    end

    if nargin == 4
        orientacao = 1 ;
    elseif orientacao == 1 | orientacao == -1
        % OK!
    else
        error( '*** ERRO *** Parametro 5 ("orientacao") inválido! Deve ser 1 ou -1!' ) ;
    end

    % dá erro se rho for negativo (exemplo: usando coordenada y no lugar)
        rho = abs( rho ) ;

    k =  4 * a .* rho ./ ( ( a + rho ) .^ 2 + z .^ 2 ) ;
    [ K , E ] = ellipke( k ) ;

    termo1 = MI_ZERO * i ./ ( 2 * pi * sqrt( ( a + rho ) .^ 2 + z .^ 2 ) ) ;
    termo2 = K + ( a .^ 2 - rho .^ 2 - z .^ 2 ) .* E ./ ...
                 ( ( a - rho ) .^ 2 + z .^ 2 ) ;

    Bz = termo1 .* termo2 * orientacao ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FIM DA FUNÇÃO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Apêndice 5: Referência bibliográfica

- J. Van Bladel, *Eletromagnetic Fields*. New York: Hemisphere Publishing, 1985 – páginas 155 e 156

Apêndice 6: Uma última foto do sistema


CAUTION