User Guide for MATLAB Relaxivity Computation Script

Ahmed Al-Shami

October 28, 2024

Contents

1 Introduction

This User Guide provides comprehensive instructions for using a MATLAB script designed to compute and plot the relaxivity $r_2(\omega_I)$ and its contributions as a function of frequency. The script models the relaxation processes in magnetic nanoparticles, considering both Néel and Brownian relaxation mechanisms, and decomposes the relaxivity into transversal, Curie, and fluctuation contributions.

2 Prerequisites

Before using the MATLAB script, ensure that you have the following:

- MATLAB installed on your computer (version R2016a or later recommended).
- Basic understanding of MATLAB operations and script execution.
- Familiarity with magnetic relaxation concepts is beneficial but not mandatory.

3 Installation

3.1 Download the Script

Save the MATLAB script provided in the previous section into a file named `relaxivity_plot.m`.

3.2 Set Up the Environment

1. Open MATLAB. 2. Navigate to the directory containing `relaxivity_plot.m` using the `Current Folder` panel or the `cd` command. For example:

```
cd 'C:\Users\YourUsername\Documents\MATLAB'
```

4 Running the Script

To execute the script:

1. Open MATLAB and ensure that the Current Folder is set to the directory containing `relaxivity_plot.m`. 2. In the Command Window, type:

```
relaxivity_plot
```

3. Press `Enter`. The script will run and generate a plot displaying the relaxivity $r_2(\omega_I)$ alongside its transversal, Curie, and fluctuation contributions across the specified frequency range.

5 Understanding the Outputs

Upon successful execution, the script produces a log-scaled plot with the following features:

- **Relaxivity** $r_2(\omega_I)$: The primary output showing how relaxivity varies with frequency.
- **Transversal Contribution**: Represents the component of relaxivity due to transversal relaxation processes.
- **Curie Term**: Corresponds to the Curie-like relaxation behavior.
- **Fluctuation Term**: Accounts for relaxivity arising from fluctuations in the magnetic moment.

The plot includes labeled axes, a legend distinguishing each component, and a grid for better readability.

6 Detailed Explanation of the Script

Below is the MATLAB script with detailed comments explaining each section and computation.

```
% MATLAB Script to Compute and Plot Relaxivity r2 vs Frequency

% Clear workspace and command window for a fresh environment
clear;
clc;

%% Constants
% Physical constants and parameters
mu0 = 4 * pi * 1e-7;               % T m/A, vacuum permeability
gamma_H = 42.577e6;                % rad/(T s), gyromagnetic ratio of proton
D = 2.3e-9;                        % m^2/s, water self-diffusion coefficient
N = 3.7e13;                        % 1/m^3, number of particles per unit volume
R = 5e-9;                          % m, particle radius
d = 0.7e-9;                        % m, dead layer thickness
R_m = R - d;                       % m, magnetic core radius
M_S = 0.52668 / mu0;               % A/m, saturation magnetization converted from T to A/m
k_B = 1.380649e-23;                % J/K, Boltzmann constant
T = 300;                           % K, temperature
eta = 0.005;                       % Pa s , viscosity of water
tau0 = 4.5e-11;                    % s, attempt time
Kn = 1.1e5;                        % J/m^3, anisotropy constant (assumed value)

%% Magnetic Moments
% Calculate the volume of the magnetic domain and the magnetic moment
V_m = (4/3) * pi * R_m^3;          % m^3, volume of the magnetic domain
mu = M_S * V_m;                    % A m^2, magnetic moment

%% External Magnetic Field
B0 = 1.5;                          % T, typical MRI field strength

%% Compute Magnetic Moment Averages
```

```
    % Calculate mean and variance of magnetic moment components using a helper function
    [mu_parallel_mean, mu_parallel_mean_squared, mu_parallel_variance, mu_perp_squared_mean] = ...
    compute_magnetic_moment_averages(mu, B0, k_B, T);

    %% Relaxation Times
    % Compute N el and Brownian relaxation times
    tau_N = tau0 * exp(Kn * V_m / (k_B * T));      % s, N el relaxation time
    tau_B = (4 * pi * eta * R^3) / (k_B * T);     % s, Brownian relaxation time

    % Effective relaxation times (assuming independent processes)
    tau_perp = 1 / (1 / tau_N + 1 / tau_B);       % s, transversal relaxation time
    tau_parallel = tau_B;                         % s, longitudinal relaxation time

    %% Sojourn Time
    tau_D = R^2 / D;                               % s

    %% Frequency Range: 0.1 MHz to 1000 MHz
    % Define a frequency range from 0.1 MHz to 1000 MHz, converted to radians per second
    omega_I = linspace(0.1e6 * 2 * pi, 1000e6 * 2 * pi, 1000); % rad/s

    %% Precompute Constants
    % Calculate a prefactor used in subsequent computations
    prefactor = ((mu0 * gamma_H)^2) / (135 * pi) * (N / (D * R));

    %% Compute Constant Term C
    % Compute constant contributions independent of frequency
    x_c_perp = tau_D / tau_perp;
    x_c_parallel = tau_D / tau_parallel;

    % Evaluate the g-function for perpendicular and parallel components
    g_c_perp = g_function(x_c_perp);
    g_c_parallel = g_function(x_c_parallel);

    % Calculate contributions to the constant term C
    C_transversal = 3 * mu_perp_squared_mean * g_c_perp;
    C_fluctuation = 4 * mu_parallel_variance * g_c_parallel;
    C_curie = 4 * mu_parallel_mean_squared;

    % Total constant term
    C = prefactor * (C_transversal + C_fluctuation + C_curie);

    %% Prepare Arrays for Results
    % Initialize arrays to store results for each frequency
    R2 = zeros(size(omega_I));
    transversal_contribution = zeros(size(omega_I));
    curie_term = zeros(size(omega_I));
    fluctuation_term = zeros(size(omega_I));

    %% Compute Frequency-Dependent Term V(omega_I)
    % Loop over each frequency to compute V and update R2 and its components
```

```
for idx = 1:length(omega_I)
omega = omega_I(idx);

% Compute complex arguments for the g-function
x_v_perp = tau_D * (1i * omega + 1 / tau_perp);
x_v_parallel = tau_D * (1i * omega + 1 / tau_parallel);
x_v_zero = tau_D * (1i * omega);

% Evaluate the g-function for each component
g_v_perp = g_function(x_v_perp);
g_v_parallel = g_function(x_v_parallel);
g_v_zero = g_function(x_v_zero);

% Calculate V components based on the theoretical model
V_transversal = (7/2) * mu_perp_squared_mean * g_v_perp;
V_fluctuation = 3 * mu_parallel_variance * g_v_parallel;
V_curie = 3 * mu_parallel_mean_squared * g_v_zero;

% Total V for the current frequency
V = V_transversal + V_fluctuation + V_curie;

% Update R2 and its individual contributions
R2(idx) = C + prefactor * real(V);
transversal_contribution(idx) = prefactor * real(V_transversal);
curie_term(idx) = prefactor * real(V_curie);
fluctuation_term(idx) = prefactor * real(V_fluctuation);
end

%% Convert R2 to Relaxivity r2
% Calculate the volume of a single particle and its concentration
V_particle = (4/3) * pi * R^3;                              % m^3, volume of a single particle
particle_concentration = N * V_particle;              % particles/m^3

% Convert concentration from particles/m^3 to mol/L
% (particles/m^3) * (1 mol / 6.022e23 particles) * (1 m^3 / 1000 L)
particle_concentration_mol_L = particle_concentration / 6.022e23 / 1e3; % mol/L

% Avoid division by zero by setting very small concentrations where necessary
particle_concentration_mol_L(particle_concentration_mol_L == 0) = 1e-12;

% Compute relaxivities by normalizing R2 and its components
r2 = R2 ./ particle_concentration_mol_L;                          % s^-1 (mol/L)^-1
transversal_contribution_r = transversal_contribution ./ particle_concentration_mol_L;
curie_term_r = curie_term ./ particle_concentration_mol_L;
fluctuation_term_r = fluctuation_term ./ particle_concentration_mol_L;

%% Plotting
% Generate a log-scaled plot of relaxivity and its contributions
figure('Position', [100, 100, 800, 600]);
```

```
    plot(omega_I / (2 * pi * 1e6), r2, 'DisplayName', 'r_2(\omega_I)', 'LineWidth', 2);
    hold on;
    plot(omega_I / (2 * pi * 1e6), transversal_contribution_r, 'DisplayName', 'Transversal Contribution', 'LineStyle', '--');
    plot(omega_I / (2 * pi * 1e6), curie_term_r, 'DisplayName', 'Curie Term', 'LineStyle', '-.');
    plot(omega_I / (2 * pi * 1e6), fluctuation_term_r, 'DisplayName', 'Fluctuation Term', 'LineStyle', ':');
    xlabel('Frequency \omega_I (MHz)', 'FontSize', 14);
    ylabel('r_2 (s^{-1}  (mol/L)^{-1})', 'FontSize', 14);
    title('Relaxivity r_2(\omega_I) and its Contributions vs Frequency', 'FontSize', 16);
    set(gca, 'XScale', 'log');              % Set x-axis to logarithmic scale
    legend('FontSize', 12);
    grid on;
    hold off;

    %% Helper Functions

    % Function to compute magnetic moment averages
    function [mu_parallel_mean, mu_parallel_mean_squared, mu_parallel_variance, mu_perp_squared_mean] = ...
    compute_magnetic_moment_averages(mu, B0, k_B, T)
    % Compute the average and variance of the longitudinal and transversal components of the magnetic moment.
    %
    % Parameters:
    %   mu (float): Magnetic moment (A m^2)
    %   B0 (float): External magnetic field (T)
    %   k_B (float): Boltzmann constant (J/K)
    %   T (float): Temperature (K)
    %
    % Returns:
    %   mu_parallel_mean (float): Mean longitudinal component
    %   mu_parallel_mean_squared (float): Mean of the squared longitudinal component
    %   mu_parallel_variance (float): Variance of the longitudinal component
    %   mu_perp_squared_mean (float): Mean of the squared transversal component

    x = mu * B0 / (k_B * T);

    % Langevin function: L(x) = coth(x) - 1/x
    if x ~= 0
    L = coth(x) - 1/x;
    else
    L = 0;
    end

    mu_parallel_mean = mu * L;                                     % Mean longitudinal component
    mu_parallel_mean_squared = mu_parallel_mean^2;                 % Mean squared longitudinal component
    mu_parallel_squared_mean = mu^2 * ((3 * L) / x - L^2);         % Mean of squared longitudinal component
```

```
    mu_parallel_variance = mu_parallel_squared_mean -
    mu_parallel_mean_squared; % Variance
    mu_perp_squared_mean = 0.5 * (mu^2 - mu_parallel_squared_mean);
                % Mean squared transversal component
    end

    % Function to compute the real part of g(x)
    function g = g_function(x)
    % Compute the real part of the function g(x) as defined in the
    theoretical model.
    %
    % Parameters:
    %   x (complex or float): Input value (can be complex)
    %
    % Returns:
    %   g (float): Real part of g(x)

    sqrt_x = sqrt(x);                                    % Compute
    square root of x
    numerator = 1 + sqrt_x / 4;                          % Numerator of
    g(x)
    denominator = 1 + sqrt_x + (4 * x) / 9 + (sqrt_x .* x) / 9; %
    Denominator of g(x)
    g = real(numerator ./ denominator);                  % Real part of
    g(x)
    end
```

Listing 1: MATLAB Script for Computing and Plotting Relaxivity $r_2(\omega_I)$

7 Script Structure and Components

The script is organized into several sections, each responsible for specific computations. Below is an overview of each section:

7.1 1. Initialization

- `clear; clc;` – Clears the MATLAB workspace and command window to ensure no residual variables interfere with the script.

7.2 2. Constants

Defines all necessary physical constants and parameters required for the calculations, such as the vacuum permeability (μ_0), gyromagnetic ratio (γ_H), diffusion coefficient (D), particle radius (R), etc.

7.3 3. Magnetic Moments

Calculates the volume of the magnetic domain (V_m) and the magnetic moment (μ) based on the saturation magnetization (M_S).

7.4 4. External Magnetic Field

Sets the external magnetic field strength (B_0), typically corresponding to a standard MRI field strength.

7.5 5. Compute Magnetic Moment Averages

Calls the helper function `compute_magnetic_moment_averages` to compute the mean and variance of the longitudinal and transversal components of the magnetic moment. These statistical measures are essential for determining relaxation contributions.

7.6 6. Relaxation Times

Calculates the Néel (τ_N) and Brownian (τ_B) relaxation times, which characterize different mechanisms by which magnetic moments relax to equilibrium.

7.7 7. Sojourn Time

Determines the sojourn time (τ_D), related to the diffusion of particles.

7.8 8. Frequency Range

Defines the range of frequencies (ω_I) over which relaxivity will be computed, spanning from 0.1 MHz to 1000 MHz, converted to radians per second.

7.9 9. Precompute Constants

Calculates a prefactor that streamlines subsequent computations, incorporating constants like μ_0 and γ_H.

7.10 10. Compute Constant Term C

Evaluates a constant term (C) that encompasses contributions independent of frequency, combining transversal, fluctuation, and Curie terms.

7.11 11. Prepare Arrays for Results

Initializes arrays to store computed values of R_2 and its individual contributions across all frequencies.

7.12 12. Compute Frequency-Dependent Term V(ω_I)

Iterates over each frequency, computing complex arguments for the g-function, evaluating the function, and aggregating the contributions to R_2.

7.13 13. Convert R2 to Relaxivity r2

Transforms the computed R_2 values into relaxivity (r_2) by accounting for particle concentration. This step ensures that relaxivity is expressed per unit concentration.

7.14 14. Plotting

Generates a log-scaled plot showcasing $r_2(\omega_I)$ and its individual contributions. The plot is enhanced with labels, a title, a legend, and grid lines for clarity.

7.15 15. Helper Functions

Includes two helper functions:

- `compute_magnetic_moment_averages`: Calculates statistical measures of the magnetic moment components using the Langevin function.
- `g_function`: Evaluates the real part of the function $g(x)$ as defined in the theoretical model.

8 Customization and Parameters

Users can modify several parameters to tailor the script to specific scenarios:

- **Physical Constants**: Adjust values like γ_H, D, N, R, etc., to match different materials or experimental conditions.
- **Temperature (T)**: Change the temperature to study its effect on relaxivity.
- **Frequency Range**: Modify `omega_I` to explore different frequency ranges.
- **Anisotropy Constant (K_n)**: Alter K_n to investigate its influence on relaxation times.

9 Troubleshooting

9.1 Common Issues

- **Division by Zero**: The script includes safeguards against division by zero by setting very small concentrations where necessary. If you encounter `NaN` or `Inf` values, ensure that input parameters are within realistic physical ranges.
- **Convergence Errors**: If the script fails to converge or produces unexpected results, verify that the physical constants and parameters are correctly defined and consistent with the units.
- **Plot Not Displaying**: Ensure that the MATLAB environment is correctly set up to display figures. Check for any errors in the Command Window that might prevent the plot from generating.

9.2 Error Messages

Review the MATLAB Command Window for specific error messages. Common errors may relate to undefined variables or incorrect function usage. Refer to the script's comments for guidance on resolving these issues.

10 Further Enhancements

Users interested in extending the script's capabilities can consider the following enhancements:

- **Vectorization**: Optimize the script by vectorizing loops for improved performance, especially for larger frequency ranges.
- **User Inputs**: Incorporate user input prompts or GUI elements to allow dynamic parameter adjustments without modifying the script.
- **Exporting Results**: Add functionality to export computed relaxivity data to external files (e.g., CSV, Excel) for further analysis.
- **Advanced Plotting**: Enhance plots with interactive features or additional customization options to better visualize complex data.

11 Conclusion

This MATLAB script provides a robust framework for computing and visualizing relaxivity $r_2(\omega_I)$ in magnetic nanoparticles. By following this user guide, you can effectively utilize and customize the script to suit various research and analysis needs in the field of magnetic relaxation.