CoMoTk User Guide

Carl Ganter

July 14, 2020

Contents

1 Configuration Model

1.1 Introduction

Any MRI sequence can be approximated by an alternating sequence of instantaneous RF pulses and time intervals with (optional) gradient activity. The evolution of the associated time-dependent microscopic magnetization density depends on tissue composition (and possibly motion). While local solutions (e.g. of the Bloch equations) can often be obtained transparently, signal localization (selective excitation, spatial encoding) usally relies on their numerical integration, without providing much physical insight.

More transparent alternatives like extended phase graphs (EPG) exist, but their scope is limited to equidistant sequences of instantaneous RF pulses (a good review of EPG has been given by Weigel [1]) and they do not include susceptibility effects.

It has been shown recently that these limitations can be overcome with a generalized version of the so-called *configuration model* (CM), (link follows) which is applicable to arbitrary sequences and their building blocks (such as selective RF pulses). While the underlying theory is microscopic, the CM works with a dedicated Fourier decomposition of the magnetization density, tailored for natural and concise inclusion of signal localization and susceptibility effects.

`CoMoTk` is a Matlab class, which implements the configuration model as described in section 1.2.

Signal localization (i.e. the transition to the voxel scale) is discussed in section 1.3.

1.2 Microscopic Scale

We consider an arbitrary sequence of instantaneous RF pulses at time points $t_0 < t_1 < t_2 < \cdots$ with flip angles $\alpha_\nu(\boldsymbol{x})$ (allowed to be zero) and phases $\varphi_\nu(\boldsymbol{x})$. Arbitrary gradients $\boldsymbol{G}(t)$ may be played out in each time separating interval $\mathcal{I}_\nu := [t_{\nu-1}, t_\nu]$ of duration $\tau_\nu := t_\nu - t_{\nu-1}$. The corresponding zero–order gradient moment vector $\boldsymbol{p}_\nu$ is defined as usual

$$\boldsymbol{p}_\nu \;:=\; \gamma \int_{t_{\nu-1}}^{t_\nu} dt\, \boldsymbol{G}(t) \tag{1}$$

An equivalence relation on the set $\{\mathcal{I}_\nu\}$ can be defined by

$$\mathcal{I}_{\nu_1} \;\sim\; \mathcal{I}_{\nu_2} \qquad \Longleftrightarrow \qquad \tau_{\nu_1} \;=\; \tau_{\nu_2} \quad \wedge \quad \boldsymbol{p}_{\nu_1} \;=\; \boldsymbol{p}_{\nu_2} \tag{2}$$

It effects a partition into d different equivalence classes $\mathcal{S}_\lambda$, where $\lambda \in \{1, \ldots, d\}$. We will refer to d as the *dimension* of the configuration model and define the mapping $\lambda(\nu)$ by the condition $\mathcal{I}_\nu \in \mathcal{S}_{\lambda(\nu)}$. Note that the gradients $\boldsymbol{G}(t)$ in the intervals $\mathcal{I}_\nu \in \mathcal{S}_\lambda$ may differ by balanced gradients, e.g. in case of phase encoding.

A *static* spin, located at $\boldsymbol{x}$ with local off-resonance frequency $\omega(\boldsymbol{x})$, will accumulate the phase[1]

$$\vartheta_\lambda \;:=\; \omega(\boldsymbol{x})\,\tau_\lambda \;-\; \boldsymbol{p}_\lambda\,\boldsymbol{x} \tag{3}$$

in any $\mathcal{I}_\nu \in \mathcal{S}_\lambda$.

Within the configuration model of dimension d, we write the magnetization vector density, immediately before or after any RF pulse, in the form

[1] Note that the notations τ_ν, $\boldsymbol{p}_\nu$ and τ_λ, $\boldsymbol{p}_\lambda$ refer to different indexing. The connection is defined by the mapping $\lambda(\nu)$.

$$\boldsymbol{m}\left(\boldsymbol{x}, t_\nu^{\pm}\right) \;\approx\; \sum_{\boldsymbol{n}\in\mathbb{Z}^d} e^{i\,[\,\omega(\boldsymbol{x})\tau_{\boldsymbol{n}} \,-\, \boldsymbol{p}_{\boldsymbol{n}}\boldsymbol{x}\,]}\, \boldsymbol{m}^{(\boldsymbol{n})}\left(t_\nu^{\pm}\right) \tag{4}$$

with

$$\tau_{\boldsymbol{n}} \;:=\; \sum_{\lambda=1}^{d} n_\lambda\, \tau_\lambda \qquad\qquad \boldsymbol{p}_{\boldsymbol{n}} \;:=\; \sum_{\lambda=1}^{d} n_\lambda\, \boldsymbol{p}_\lambda \tag{5}$$

We will refer to $\boldsymbol{m}^{(\boldsymbol{n})}$ as *configuration vector* and $\boldsymbol{n}$ as *configuration order*.

1.3 Voxel Scale

At first glance, the configuration model (4) is just a cumbersome and redundant way to rewrite the microscopic magnetization density $\boldsymbol{m}\,(\boldsymbol{x})$.

The situation changes beyond the microscopic scale, in particular for the reconstructed *voxel* m_ρ, when its true value becomes apparent.

Actually, two fundamental mechanisms are available for signal localization:

- Selective Excitation
- Spatial Encoding

Both rely on the application of gradients, effecting more or less complicated signal modulations in the vininity of a given location $\boldsymbol{x}$. Under the assumption that other causes[2] for these variations can be neglected on the voxel scale, the superposition of configurations in Eq. (4) encodes just this information via the phase factors $e^{i\boldsymbol{n}\boldsymbol{\vartheta}(\boldsymbol{x})}$.

As an example, how to apply this knowledge for both localization mechanisms, let us consider some 2D sequence with Cartesian sampling and slice selective excitation. The finite readout duration is neglected and we assume the coordinate axes to be aligned parallel to the reconstructed image, such that $x_{1,2}$ encode the locations along the orthonormal in-plane vectors $\boldsymbol{e}_{1,2}$ and x_3 the position along the slice normal $\boldsymbol{e}_3$.

1.4 Spatial encoding

Due to finite sampling, the discrete reconstructed voxel signal m_ρ at position $\boldsymbol{x}_\rho$ results from convolution of the transverse magnetization density $m\,(\boldsymbol{x})$ with a *point spread function* $\phi\,(\boldsymbol{x})$

$$m_\rho \;\propto\; \phi * m\,(\boldsymbol{x}_\rho) \tag{6}$$

which in our example is just a scaled sinc function[3]

$$\phi\,(\boldsymbol{x}) \;:=\; \prod_{j=1}^{2} \operatorname{sinc}\left(\frac{x_j}{\Delta x_j}\right) \tag{7}$$

where Δx_j denotes the pixel resolution in direction j.

[2] Tissue properties, partial volume effects, B_0 inhomogeneity (beyond suceptibility variations), to name a few.
[3] We assume the normalized version: $\operatorname{sinc}(x) := \sin \pi x / \pi x$

We now insert the configuration model (4) into (6) with $f^{(\boldsymbol{n})} := e^{i\,\omega\tau_{\boldsymbol{n}}}\, m^{(\boldsymbol{n})}$ and get after short calculation

$$m_\rho \;\propto\; \sum_{\boldsymbol{n}\in\mathbb{Z}^d} \int d\boldsymbol{k}\, e^{i\,\boldsymbol{k}\boldsymbol{x}_\rho}\, \hat{f}^{(\boldsymbol{n})}\,(\boldsymbol{k}) \cdot \prod_{j=1}^{2} u\left(\frac{\pi}{\Delta x_j} - |k_j + p_{\boldsymbol{n},j}|\right) \tag{8}$$

where u is the unit step function.

If the support of $\hat{f}^{(\boldsymbol{n})}$ in direction j is approximately[4] bounded by $\pi/\Delta x_j$, only configurations with

$$|\,p_{\boldsymbol{n},j}| \;<\; \frac{2\pi}{\Delta x_j} \tag{9}$$

contribute to m_ρ.

Essentially, this matches the often encountered statement that crusher gradients should effect a 2π dephasing over the voxel dimension, in order to be effective. But we also see that this is only approximately true and depends on the severity of partial volume effects.

To summarize, $\boldsymbol{p}_{\mathbf{n}}$ provides information, which configurations $m^{(\boldsymbol{n})}$ contribute to the reconstructed signal m_ρ.

1.5 Selective excitation

In real acquisitions, selective excitation is performed via application of a resonant $\boldsymbol{B}_1^+$ field in presence of slice encoding gradients. For example, an amplitude modulated RF pulse is usually split into a few hundred intervals of equal duration, within each of which the gradient and $\boldsymbol{B}_1^+$ are held constant[5]. Since rotations around different axes do not commute in general, RF pulse design treats RF pulses as a series of instantaneous small tip angle pulses interleaved by short intervals[6] τ of constant gradient moments $\boldsymbol{p}$ in direction of the slice normal $\boldsymbol{e}_3$, cf. [2]. The accuracy of this approximate description is determined by the choice of τ.

For the reconstructed voxel m_ρ, Eq. (6) still applies in principle, since it also includes an unbounded integral along the slice normal $\boldsymbol{e}_3$. Due to the approximate description of the RF pulse, however, the configuration model according to Eq. (4) becomes periodic in this direction

$$m\,(\boldsymbol{x}) \;\equiv\; m\left(\boldsymbol{x} + n \cdot \frac{2\pi}{p} \cdot \boldsymbol{e}_3\right) \qquad\qquad n \in \mathbb{Z} \tag{10}$$

and the x_3 integral should therefore be restricted to the interval $[-\pi/p, \pi/p]$. In presence of several pulses with possibly different p_μ, the *excitation* pulse[7] defines the integration interval. In general, $\boldsymbol{p}$ is not the only gradient mit a nonzero component in direction of $\boldsymbol{e}_3$. For example, to restore spin coherence across the slice, subsequent time intervals include a rephasing moment parallel to $\boldsymbol{e}_3$.

The effect of selective excitation can therefore be handled by a weighting factor $w_{\boldsymbol{n}}$ in Eq. (4)

$$e^{-\,i\,\boldsymbol{p}_{\boldsymbol{n}}\boldsymbol{x}} \cdot m^{(\boldsymbol{n})} \;\rightarrow\; w_{\boldsymbol{n}} \cdot m^{(\boldsymbol{n})} \tag{11}$$

[4] This means that the reconstructed resolution recovers the essential fine structure of the sample. In a strict sense, $\hat{f}^{(\boldsymbol{n})}$ is not bouded at all, as it is the Fourier transform of a bounded function.

[5] VERSE pulses are an exception.

[6] In the configuration model, the small intervals generate a separate dimension with a unique index, say, $\mu = 1$ and the assignments $\tau_1 = \tau$ and $\boldsymbol{p}_1 = \boldsymbol{p}$.

[7] This should handle virtually all relevant cases: For (T)SE sequences, the excitation pulse is uniquely defined, since excitation due to imperfect refocusing cannot enter the signal (these paths are suppressed by the crusher gradients). On the other hand, every pulse in an SSFP sequence can excite spins, but here the RF pulses are usually identical (at least with respect to duration and gradient moment).

where

$$w_{\boldsymbol{n}} \;:=\; \frac{p}{2\pi} \cdot \int_{-\pi/p}^{\pi/p} dx_3 \, e^{-i\, p_{\boldsymbol{n},3} x_3} \;=\; \mathrm{sinc}\left(\frac{p_{\boldsymbol{n},3}}{p}\right) \tag{12}$$

Since p is rather small, it is sometimes possible to assume that the third component of all $\boldsymbol{p}_\mu$ is some integer multiple of p. In this case, Eq. (12) simplifies to

$$w_{\boldsymbol{n}} \;=\; \begin{cases} 1 & : \quad p_{\boldsymbol{n},3} = 0 \\ 0 & : \quad p_{\boldsymbol{n},3} \neq 0 \end{cases} \tag{13}$$

If, additionally, all nonzero in-plane gradient moments refer to (large enough) crusher gradients, the restrictions due to spatial encoding and selective excitation can be combined in a single, handy statement:

$$w_{\boldsymbol{n}} \;=\; \begin{cases} 1 & : \quad \boldsymbol{p}_{\boldsymbol{n}} = \boldsymbol{0} \\ 0 & : \quad \boldsymbol{p}_{\boldsymbol{n}} \neq \boldsymbol{0} \end{cases} \tag{14}$$

1.6 Susceptibility effects

Susceptibility related intra-voxel frequency variations $\omega_s(\boldsymbol{x})$, as part of the local resonance frequency $\omega(\boldsymbol{x})$, are typically considered as independent of gradient induced frequency modulations in the integral (6).

Under this assumption, the fluctuations are assumed to be distributed according to some zero-mean density $p(\omega_s)$ within the voxel. The associated damping factor depends on the configuration (via $\tau_{\boldsymbol{n}}$) and is given by

$$\hat{p}(\tau_{\boldsymbol{n}}) \;:=\; \int d\omega_s \, e^{i\,\omega_s \tau_{\boldsymbol{n}}} \cdot p(\omega_s) \tag{15}$$

To calculate m_ρ, we replace $m^{(\boldsymbol{n})} \to \hat{p}(\tau_{\boldsymbol{n}}) \cdot m^{(\boldsymbol{n})}$ in the configuration model (4) for all contributing $\boldsymbol{n}$.

Most commonly, a Lorentzian distribution $p(\omega_s)$ is assumed and the damping factor $\hat{p}(\tau_{\boldsymbol{n}})$ takes the familiar form

$$\hat{p}(\tau_{\boldsymbol{n}}) \;=\; e^{-R_2' \tau_{\boldsymbol{n}}} \tag{16}$$

2 Usage

To familiarize with the toolkit, the scripts in the `test` and `examples` folders are recommended as a good starting point. Following, a brief overview of the basic functionality.

2.1 Initialize CoMoTk

First, we need to create an instance of CoMoTk:

```
cm = CoMoTk;
```

Next, we have to setup a few mandatory tissue parameters. In the simpliest case of a 1-peak model without diffusion, this can look like this:

```
cm.R1 = 0.01;       % longitudinal relaxation rate
cm.R2 = 0.1;        % transverse relaxation rate
cm.D = 0;           % apparent diffusion coefficient
```

It is also possible, to specify more complex n-peak models with variable relative weighting ($\propto$ proton density), chemical shift and diffusivity. Here, a 2-peak example:

```
cm.R1 = [ 0.01, 0.02 ];
cm.R2 = [ 0.1, 0.2 ];
cm.D = [ 3, 1.5 ];
cm.w = [ 0.6, 0.4 ];    % relative weight (does not need to add to 1)
cm.dom = [ 0, -0.1 ];   % chemical shift (angular frequency)
```

Setting `w` and `dom` is optional. If not set, they default to 1 and 0, respectively.

Another optional variable is the relative B_1^+ field (default = 1):

```
cm.B1 = 0.8;
```

There are further options, for which it is possible to modify the default settings. The most important one is the desired accuracy. A nonzero value of `epsilon` is set, if the number of stored configurations needs to be restricted, e.g. due to memory or performance restrictions.

```
options = cm.options;    % get default options

options.alloc_d = 3;     % allocated number of dimensions
options.alloc_n = 10000; % allocated number of configurations
options.epsilon = 0;     % for maximal accuracy (enough memory needed)
options.verbose = true;  % for more output
options.debug = true;    % for debugging purposes (look into CoMoTk.m)

cm.options = options;    % activate new options
```

Now we have to specify the initial configuration vector, corresponding to $\boldsymbol{n} = 0$. It is supplied as a real vector (row or column) in the usual convention (m_x, m_y, m_z):

```
cm.init_configuration ( [ 0; 0; 1 ] );  % longitudinal magnetization
```

In addition to $m_\nu^{(\boldsymbol{n})}$, knowledge of certain partial derivatives $\partial m_\nu^{(\boldsymbol{n})}/\partial\xi$ is sometimes desired as well, e.g. for numerical optimization. This is supported in CoMoTk for $\xi \in \{R_1, R_2, D, B_1, \alpha_\mu, \varphi_\mu, \tau_\mu, \boldsymbol{p}_\mu, \boldsymbol{s}_\mu, \boldsymbol{S}_\mu\}$. The following command (a full example, involving every possible variable) is used, to inform CoMoTk, which derivatives need to be calculated:[8]

```
  cm.set_derivatives ( ...
'R1', [ 1, 3 ], ...           % tissue handles (n-peak model)
'R2', 2, ...                  % for a 1-peak model the handle (= 1)
'D', [ 2, 3 ], ...            % must be supplied as well
'B1', ...                     % only B1 has no handle
'FlipAngle', [ 2, 3 ], ...    % non-tissue handles can be chosen
'Phase', [ 2, 1004, 12 ], ... % arbitrarily
'tau', [ 3, 4, 6 ], ...       % time interval (= \mu)
'p', [ 1, 2, 4 ], ...         % gradient moment (see below)
```

[8] Currently, CoMoTk supports isotropic diffusion only. Therefore, the gradient shape is determined by $\mathtt{s} := (\boldsymbol{s}_\mu, \mathrm{tr}\,(\boldsymbol{S}_\mu))$. Derivatives with respect to `s` for a given interval μ make only sense, if the shape does not change.

```
's', [ 4, 7 ] ...              % gradient shape (see below)
);
```

Of course, only the desired subset of parameter/handle combinations needs to be supplied. If the command is not given, no derivatives are calculated by default.

After having initialized everything, we proceed with the actual sequence. Here, we apply instantaneous RF pulses and time intervals, typically in an alternate fashion.[9]

2.2 RF pulse

Calling an RF pulse is as simple as:

```
cm.RF( flip, phase );  % RF pulse with flip angle and phase [rad]
```

If partial derivatives with respect to flip angle(s) and/or phase(s) have been set before, the associated handles can be additionally supplied according to the following format:[10]

```
cm.RF( flip, phase, 'FlipAngle', flip_handle, 'Phase', phase_handle );
```

2.3 Time interval

Whether the time derivatives are needed or not, executing the time interval always needs specification of the (otherwise arbitrary) index `mu` $:= \mu$, which is defined as in section 1.2. At least in the first call of each distinct interval, the duration `tau` $:= \tau_\mu$ must be supplied also:

```
cm.time( mu, 'tau', tau );
```

If we also require the gradient moment `p` $:= \boldsymbol{p}_\mu$ for the simulations (diffusion effects) or the results (e.g. slice profile), we have to add this parameter as well:[11]

```
cm.time( mu, 'tau', tau, 'p', p );
```

Called like this, a constant gradient shape according to (??) and (??) is assumed.

For arbitrary shapes, the variable `s` $:= (\boldsymbol{s}_\nu, \mathrm{tr}\,(\boldsymbol{S}_\nu))$ must be added too:[12]

```
cm.time( mu, 'tau', tau, 'p', p, 's', s );
```

In later calls, only the handle is required:[13]

```
cm.time( mu );
```

Note, however, that this shorthand form is not safe for nonzero `epsilon`, since the dimension `mu` could have been eliminated (together with any knowledge about `tau` and `p`) by a previous call of `meltdown()`. In case of doubt, always specify all parameters.

2.4 Get results

After any RF pulse or time interval, we can obtain the sum[14] (4) like this:

```
res = cm.sum( param );
```

`param` is a structure with optional fields:

[9] There are no restrictions, though, i.e. multiple subsequent RF pulses or time intervals are allowed as well.

[10] If only the flip angle derivative is needed, the phase handle is not required (and vice versa). The user is responsible for the validity of value/handle combinations. Otherwise, the result is unpredicted.

[11] `p` must be a 3×1 or 1×3 array. If we set an element of `p` equal to `Inf`, it is interpreted as an ideal spoiler, as defined in section 4.2.

[12] `s` must be a 4×1 or 1×4 array.

[13] The full version is also allowed. Of course, `s` still needs to be supplied, if the shape is variable.

[14] For the isochromat the sum over all tissues (if there is more than one) is performed as well.

`omega` = Local angular off-resonance frequency $\omega(\boldsymbol{x})$

`x` = Position $\boldsymbol{x}$ according to the definition (3)

`b_n` = Subset from the set of stored configuration orders, $S_\nu^\pm$, to be included in the result.

`w_n` = explicit weighting factors (`length( w_n ) = sum( b_n )`)

For unset fields, the following defaults are assumed:

- `omega` $= 0$
- `x` $= 0$
- `b_n` = `cm.b_n` (= whole sum in Eq. (4))
- `w_n` $= 1$

The result is returned separately as transverse[15] (`res.xy`) and longitudinal (`res.z`) component. Calculated derivatives with respect to `X` are returned as `res.dm_dX.xy` and `res.dm_dX.z`, where `X` is any member of the set $\{$`R1`, `R2`, `D`, `B1`, `FlipAngle`, `Phase`, `tau`, `p`, `s`$\}$. We have `size(res.dm_dX) = [a,b]`, where `a = 1`, except for `X = p` with `a = 3` and `X = s` with `a = 4`. The second dimension `b` is just the number of derivatives to be calculated for each parameter as defined in `set_derivatives()`.

For a restriction of the summation, the `find` method can be used to generate `b_n`:

```
b_n = cm.find( mu, n );
```

If `mu` and `n` are scalars, we have `b_n` $= \{\boldsymbol{n} \in S_\nu^\pm : n_{\texttt{mu}} = \texttt{n}\}$. If no matching configurations are found, `b_n = []` is returned.

`mu` and `n` can also be 1d-arrays of equal length. This can be used as a shorthand for a combination with the `|` operator. For example, we obtain for arrays of length 3 a result equivalent to

```
  b_n = ...
cm.find( mu( 1 ), n( 1 ) ) | ...
cm.find( mu( 2 ), n( 2 ) ) | ...
cm.find( mu( 3 ), n( 3 ) );
```

If we want to single out a specific configuration $\boldsymbol{n} \in S_\nu^\pm$, we have to use the `&` operator explicitly. For example, we get for $d = 3$:

```
  b_n = ...
cm.find( cm.mu( 1 ), n( 1 ) ) & ...
cm.find( cm.mu( 2 ), n( 2 ) ) & ...
cm.find( cm.mu( 3 ), n( 3 ) ) );
```

It is also possible, to restrict the mask `b_n` directly. For example, to extract all stored configurations, which satisfy $\boldsymbol{p}_{\boldsymbol{n}} = \boldsymbol{0}$, cf. Eq. (14), one could use

```
b_n = cm.b_n & reshape( ~any( cm.p_n ), size( cm.b_n ) );
```

3 Class Reference

3.1 Properties (selection)

3.1.1 `n_tissues`

info: number of subspecies

format: integer (≥ 1)

[15] In the traditional interpretation `res.xy` $= m_x + i\,m_y = \sqrt{2} \cdot m_1$, where the last term corresponds to our chosen notation in Eq. (??).

3.2 Preparation

3.2.1 `CoMoTk`

action: constructor, creates an instance `cm` of `CoMoTk`

call: `cm = CoMoTk`

input: none

output: instance of class

3.2.2 `set.R1`

action: set relaxation rate R_1 for all tissue subspecies

mandatory: yes

call: `cm.R1 = R1`

input: `R1` is a vector (row or column). `length( R1 )` == `n_tissues`

output: none

3.2.3 `get.R1`

action: get R_1 value(s)

call: `cm.R1`

input: none

output: (row) vector of R_1 of all subspecies

3.2.4 `set.R2`

action: set relaxation rate R_2 for all tissue subspecies

mandatory: yes

call: `cm.R2 = R2`

input: `R2` is a vector (row or column). `length( R2 )` == `n_tissues`

output: none

3.2.5 `get.R2`

action: get R_2 value(s)

call: `cm.R2`

input: none

output: (row) vector of R_2 of all subspecies

3.2.6 `set.D`

action: set apparent diffusion coefficient D (scalar or tensor) for all tissue subspecies

mandatory: yes

call: `cm.D = D`

input: `D` is the apparent diffusion coefficient. `size( D )` must be either `[ 1, n_tissues ]` (isotropic diffusion coefficient) or `[ 3, 3, n_tissues ]` (diffusion tensor)

output: none

3.2.7 `get.D`

action: get apparent diffusion coefficient/tensor

call: `cm.D`

input: none

output: apparent diffusion coefficient/tensor, `size( cm.D )` == `[ 1, n_tissues ]` (isotropic diffusion) or `[ 3, 3, n_tissues ]` (anisotropic diffusion)

3.2.8 `set.B1`

action: set relative B_1

mandatory: no (default = 1)

call: `cm.B1 = B1`

input: `B1` is a scalar ≥ 0

output: none

3.2.9 `get.B1`

action: get B_1 value

call: `cm.B1`

input: none

output: B_1 value

3.2.10 `set.mu`

action: set relative proton density for all tissue subspecies

mandatory: no (default = `ones( 1, n_tissues )`)

call: `cm.mu = mu`

input: `mu` is a vector (row or column). `length( mu )` == `n_tissues`

output: none

3.2.11 `get.mu`

action: get relative proton density value(s)

call: `cm.mu`

input: none

output: (row) vector of proton densities of all subspecies

3.2.12 `set.dom`

action: set chemical shift frequency $\delta\omega$ for all subspecies

mandatory: no (default = `zeros( n_tissues, 1 )`)

call: `cm.dom = dom`

input: `dom` is a vector (row or column). `length( dom )` == `n_tissues`

output: none

3.2.13 `get.dom`

action: get chemical shift frequency of all tissue subspecies

call: `cm.dom`

input: none

output: (column) vector of chemical shift frequencies

3.2.14 `set.R2p`

action: set decay rate R_2' according to Lorentzian distribution for all tissue subspecies

mandatory: no (default = 0)

call: `cm.R2p = R2p`

input: `R2p` is a vector (row or column). `length( R2p )` == `n_tissues`

output: none

3.2.15 `get.R2p`

action: get R_2' value(s)

call: `cm.R2p`

input: none

output: (column) vector of R_2' of all subspecies

3.2.16 `set.k`

action: set the matrix of transition rates for magnetization exchange.

mandatory: no (default = [])

call: `cm.k = k`

input: `k` is a matrix. `size( k ) == [ n_tissues, n_tissues ]`. The element `k( i, j ) ≥ 0` is the rate of the transition `j` → `i`. Diagonal elements are ignored and calculated according to the sum rule for particle conservation as explained in the article (citation). For consistency with the equilibrium proton density, the condition `cm.k * cm.mu' ≡ 0` must also be satisfied. (the matrix `cm.k` has the correct diagonal elements, see below) Note that this condition is *not* checked by `CoMoTk` and has to be guaranteed by the user!

output: none

3.2.17 `get.k`

action: get the transition matrix k

call: `cm.k`

input: none

output: transition matrix k with calculated diagonal elements

3.2.18 `set.options`

action: set various options

mandatory: no (default settings exist)

call: `cm.options = options`

input: `options` is a structure with the following (mandatory) elements (hint: use the `get.options` method to get a proper structure with default values and modify the parameters, you want to change):

`alloc_d`: predicted dimension of CM to avoid reallocation (default = 1). program should also work fine, if set too low

`alloc_n`: predicted number of occupied states of CM to avoid reallocation (default = 1000). program should also work fine, if set too low

`epsilon`: discard configuration vectors with L2 norm smaller than this to prevent memory overflow and to speed up the simulation. use only, if necessary. (default = 0, i.e. nothing is discarded)

`rapid_meltdown`: discarding configurations should be faster (not guaranteed though) but requires more memory (default = `true`)

`verbose`: a bit more output such as actual number of occupied configurations (default = `false`)

`debug`: stores information for debugging purposes (default = `false`)

output: none

3.2.19 `get.options`

action: get program options

call: `cm.options`

input: none

output: cf. input of `set.options`

3.2.20 `init_configuration`

action: specify initial magnetization of all subspecies and prepare everything. should be called *after* any `set.*` method. (only `set_derivatives` can and must be called later)

mandatory: yes

call: `cm.init_configuration( m )`

input: `m` is the initial state with `size( m ) = [ 3, n_tissues ]`. it corrsponds to the zero order ($\boldsymbol{n} = \mathbf{0}$) configuration, which is the only occupied state in absence of prior RF pulses.

output: none

3.2.21 `set_derivatives`

action: define the derivatives to be calculated

mandatory: no

call: `cm.set_derivatives( param )`

input: `param` is a structure with one or more fields `X` $\in$ `{ R1, R2, D, B1, FlipAngle, Phase, tau, p, s, S }`. The supplied information is situational:

- `X` $\in$ `{ R1, R2, D }`
  - $1 \leq$ `size( X )` $\leq$ `n_tissues`
  - `X( j )` $\in$ `{ 1, ..., n_tissues }` sets the subspecies, for which the derivatives are calculated.
- `X` = `B1`
  - value unimportant
  - derivatives with respect to `B1` will be calculated, if field exists
- `X` $\in$ `{ FlipAngle, Phase }`
  - `X( j )` $\in$ `{ supplied handles }` (see method `RF`)
- `X` $\in$ `{ tau, p, s, S }`
  - `X( j )` $\in$ `{ cm.lambda }` (see method `time`)

output: none (see the `sum` method below for details about how the calculated derivatives can be accessed)

3.3 Execution

3.4 Results

4 Design Notes

The Matlab class `CoMoTk` implements the configuration model according to its definition in section 1.2. In addition to executing arbitrary sequences, various first order partial derivatives (with respect sequence and tissue parameters) may be extracted as well.

4.1 Units

The toolkit assumes the following units throughout:

- **time:** [ms]
- **relaxation rate:** [1/ms]
- **angular frequency:** [rad/ms]
- **apparent diffusion coefficient:** [μm^2/ms]
- **angles:** [rad]
- **length, position:** [μm]
- **gradient moment, spatial frequency:** [rad/μm]
- **velocity:** [mm/s]

4.2 Spoiler gradients

Spoiler and crusher gradients can be differentiated by their intended action. While the latter are used in sequences for de– *and* rephasing, the task of the former is to destroy transverse coherences reliably [3]. In principle, this goal can be (partly) reached by two effects:

- **Voxel scale:** Suppression of unwanted configurations via crusher gradients (included in $p_{\boldsymbol{n}}$), as outlined in section 1.4.
- **Microscopic scale:** By diffusion effects.

With respect to the second mechanism, it follows from Eq. (??) and the recursion (??) that ideal spoiling can be accomplished in the limit

$$\operatorname{tr}\left(\boldsymbol{D}\,\boldsymbol{S}_\nu\right) \;\to\; \infty \tag{17}$$

since then we have

$$\boldsymbol{F}^{(\boldsymbol{n})}\left(\tau_\mu\right) \;\to\; \begin{pmatrix} 0 & & \\ & e^{-\,\tau_\mu\,\boldsymbol{p}_{\boldsymbol{n}}^T \boldsymbol{D}\,\boldsymbol{p}_{\boldsymbol{n}}} & \\ & & 0 \end{pmatrix} \tag{18}$$

and transverse magnetization is eliminated in all configuration orders.

Note that the condition (17) does not necessarily[16] require $p_\mu \to \infty$ (or $p_{\boldsymbol{n}} \to \infty$). On the other hand, $p_\mu \to \infty$ will usually require[17] (17) as well.

4.3 General properties of occupied configurations

Proposition 1. *In absence of magnetization transfer and for $T_2 \leq T_1$, we always have*

$$\sum_{\boldsymbol{n}} \left\| \boldsymbol{m}^{(\boldsymbol{n})} \right\|_2^2 \;\leq\; m_{eq}^2 \tag{19}$$

[16] Strong balanced gradients with $p_\mu = 0$ are a simple counterexample. In the specific case (??), however, the conditions (17) and $p_\mu \to \infty$ are equivalent.

[17] More accurately, if we replace "∞" by "large", this follows from the actual limitations of gradient hardware.

Proof. The RF pulses do not affect $\left\|\boldsymbol{m}^{(\boldsymbol{n})}\right\|$. Therefore we only need to consider the time intervals and the proof can be done by induction. From (??), we derive

$$\left|m_{+,j}^{(\boldsymbol{n})}\right| \;=\; \begin{cases} \left|F_{\nu,0}^{(\boldsymbol{0})}\, E_{00}\, m_{-,0}^{(\boldsymbol{0})} + (1-E_1)\, m_{eq}\right| & : \quad \boldsymbol{n}=\boldsymbol{0} \;\wedge\; j=0 \\ \left|F_{\nu,j}^{(\boldsymbol{n}-j\boldsymbol{e}_\mu)}\, E_{jj}\right| \left|m_{-,j}^{(\boldsymbol{n}-j\boldsymbol{e}_\mu)}\right| & : \quad \text{else} \end{cases} \tag{20}$$

In combination with $T_2 \le T_1$ and assuming that (19) holds for $\boldsymbol{m}_-^{(\boldsymbol{n})}$, we therefore obtain

$$\begin{aligned} \sum_{\boldsymbol{n}} \left\|\boldsymbol{m}_+^{(\boldsymbol{n})}\right\|_2^2 \;&\le\; E_1^2 \cdot \sum_{\boldsymbol{n}} \left\|\boldsymbol{m}_-^{(\boldsymbol{n})}\right\|_2^2 \;+\; 2\,E_1\,(1-E_1)\left|m_{-,0}^{(\boldsymbol{0})}\right| m_{eq} \;+\; (1-E_1)^2\, m_{eq}^2 \\ &\le\; \left(E_1^2 \;+\; 2\,E_1\,(1-E_1) \;+\; (1-E_1)^2\right) m_{eq}^2 \\ &=\; m_{eq}^2 \end{aligned} \tag{21}$$

which completes the proof. □

4.4 Storage considerations

The actual state of the configuration is defined by the set $\{\boldsymbol{n}\}$ of occupied configurations and their associated configuration vectors $\boldsymbol{m}^{(\boldsymbol{n})}$.

The size of the set, n_c, will increase with every time interval

$$n_c \;\to\; \begin{cases} n_c + 2d & : \quad d \to d \\ 3n_c & : \quad d \to d+1 \end{cases} \tag{22}$$

The first case applies, if the same time interval μ has been played out before[18], whereas the second case applies for new intervals[19].

Eq. (22) implies that n_c becomes particularly large for high dimensional configuration models. The most extreme scenario is a fingerprinting-type excitation pattern with distinct random intervals τ_μ. This pattern generates $n_c = 3^n$ occupied configurations after $n = d$ time intervals, easily exceeding the memory of actual computer hardware after less than about $n \approx 20$ intervals.

Fortunately, due to relaxation, magnetization forgets about its initial state with the consequence that many (if not most) of the occupied configuration vectors $\boldsymbol{m}^{(\boldsymbol{n})}$ may be safely ignored. Formally, this immediately follows from the upper bound on $\sum_{\boldsymbol{n}} \left\|\boldsymbol{m}^{(\boldsymbol{n})}\right\|_2^2$, derived in Proposition 1.

Based on some given, user-defined limit ε, we will therefore repeatedly discard all configuration vectors with $\left\|\boldsymbol{m}^{(\boldsymbol{n})}\right\|_2 < \varepsilon$. To this end, we invoke the method `meltdown()` after every time interval.

For this to work properly, the actual set $\boldsymbol{m}^{(\boldsymbol{n})}$ is stored in a two-dimensional array `m` of size[20] $3 \times n_a$, where n_a specifies the allocated (not necessarily occupied) space. This storage concept requires a considerable amount of bookkeeping, which is described in more detail in section 4.5.

To improve performance further, reallocations[21] and redundant computations[22] are minimized as well.

[18] In this case, the dimension of the configuration model, d remains unchanged.
[19] Here, the dimension d is increased by 1.
[20] The first dimension refers to the three vector components.
[21] Mainly achieved via a sufficiently large initial value of n_a.
[22] Some quantities, which are repeatedly needed in calls of identical RF pulses or time intervals are stored for reuse. When possible, updates are limited to newly entering configurations.

4.5 Bookkeeping

The `n_conf` *stored* configuration orders $\boldsymbol{n}$ of dimension `d` are collected in the $n_a \times \mu_a$ array `n`. The associated location in the allocated storage is determined by the logical $n_a \times 1$ array `b_n` and the $1 \times \mu_a$ array `b_mu`.[23]

The `d` distinct dimensions μ are stored in the $1 \times \mu_a$ array `mu`.

Prior to and after the ν^{th} time interval, we denote the set of *stored* configurations $\boldsymbol{n}$ by S_ν^- and S_ν^+, respectively. To implement the time intervals properly, particularly recursions like (**??**), it is crucial to guarantee that $S_\nu^\pm$ does not contain any holes: If $\boldsymbol{n} \in S_\nu^\pm$ and $\boldsymbol{n} + c \cdot \boldsymbol{e}_\mu \in S_\nu^\pm$ for some $\boldsymbol{n} \in \mathbb{Z}^d$ and $c \in \mathbb{Z} > 0$, we must have $\boldsymbol{n} + b \cdot \boldsymbol{e}_\mu \in S_\nu^\pm$ for every $1 \leq b \in \mathbb{Z} < c$.

Since RF pulses do not change the set of occupied configurations, we set $S_{\nu+1}^- = S_\nu^+$.

For each $\boldsymbol{n} \in S_\nu^-$ and each dimension μ, the information whether $\boldsymbol{n} \pm \boldsymbol{e}_\mu \in S_\nu^-$ is fulfilled, is stored in a separate variable.[24] In case of $\boldsymbol{n} \pm \boldsymbol{e}_\mu \in S_\nu^-$, the corresponding location in memory must be available as well.[25]

Due to the method `meltdown()`, the `n_conf` $\leq n_c$ *stored* configurations are some subset of $\mathbb{Z}^{\mathtt{d}}$ with[26] `d` $\leq d$. We demand that the stored set remains connected along each dimension, i.e. free of holes, as described above. We also demand that $\boldsymbol{n} = \boldsymbol{0}$ is included at all times.

References

[1] Matthias Weigel. Extended phase graphs: Dephasing, RF pulses, and echoes - pure and simple. *J. Magn. Reson. Imaging*, 41(2):266–295, April 2014.

[2] J. Pauly, P. Le Roux, D. Nishimura, and A. Macovski. Parameter relations for the shinnar-Le roux selective excitation pulse design algorithm (NMR imaging). *IEEE Trans. Med. Imaging*, 10(1):53–65, March 1991.

[3] Xiaohong Joe Zhou Matt A. Bernstein, Kevin F. King. *Handbook of MRI Pulse Sequences*. Elsevier, 2004.

[23] Obvioulsly, `sum( b_n )` and `sum( b_mu )` must be equal to `n_conf` and `d`, respectively.

[24] We define two logical $n_a \times \mu_a$ arrays `b_up_free` and `b_do_free` as follows: For each stored configuration $\boldsymbol{n}$ and dimension μ, the associated entry in `b_up_free` is `true`, if the direct neighbor $\boldsymbol{n} + \boldsymbol{e}_\mu$ is *not* stored and `false` otherwise. Similarly, `b_do_free` is `true`, if $\boldsymbol{n} - \boldsymbol{e}_\mu$ is *not* stored and `false` otherwise.

[25] Similarly, we define two $n_a \times \mu_a$ arrays `idx_up` and `idx_do` as follows: For each stored configuration $\boldsymbol{n}$ and dimension μ, with the associated entry in `b_up_free` equal to `false`, the corresponding value `idx_up` gives the index (with respect to the first dimenasion of size n_a) of the direct neighbor $\boldsymbol{n} + \boldsymbol{e}_\mu$. For the other direction, $\boldsymbol{n} - \boldsymbol{e}_\mu$, the array `idx_do` is defined in complete analogy.

[26] A populated dimension μ, for which no configuration orders $n_\mu \neq 0$ are stored, is de facto nonexistent and can be discarded. This can, for example, happen, if the associated time period τ_μ has not been played out for a long time.