CoMoTk Matlab User Guide

Carl Ganter

August 5, 2020

In addition to the comments in the code and the provided (highly recommended) examples, this short document contains some additional tips, how to use the configuration model (CM) toolkit `CoMoTk`. It does **not** explain the underlying CM formalism, which is done extensively in a separate article (link will be provided, as soon as it becomes accessible).

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 proton density, chemical shift and diffusivity (which can also be supplied as a tensor in case of anisotropic diffusion).

Here, a 2-peak example:

```
cm.R1 = [ 0.01, 0.02 ];
cm.R2 = [ 0.1, 0.2 ];
cm.D = [ 3, 1.5 ];
cm.mu = [ 0.6, 0.4 ];     % relative proton density (does not need to add to 1)
cm.dom = [ 0, -0.1 ];     % chemical shift (angular frequency)
cm.k = [ -2, 3; 2, -3 ]; % transition rates for magnetization transfer/exchange
```

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

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

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

There are further options to modify the default settings. The most important one is the desired accuracy `epsilon`. A nonzero value restricts the number of stored configurations (at the cost of accuracy), which can be reasonable to prevent memory overflow and/or to reduce the computation time.

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

options.alloc_d = 3;            % allocated number of dimensions
options.alloc_n = 10000;        % allocated number of configurations
options.epsilon = 1e-4;         % trade accuracy against speed and/or memory
options.rapid_meltdown = false; % choice of acceleration technique
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 configuration order $\boldsymbol{n} = 0$. 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 the actual state (expressed by all the configuration vectors), knowledge of certain partial derivatives is sometimes desired as well, e.g. for numerical optimization. This is supported in `CoMoTk` for several tissue and sequence parameters. To get familiar with this concept, the best strategy may be to first look into the provided script `test_derivatives.m` and for more details into the class definition `CoMoTk.m`.

After having initialized everything, we proceed with the actual sequence.

2 Execute Sequence

2.1 RF pulse

Executing an (instantaneous) RF pulse is as simple as:

```
param = [];
param.FlipAngle = pi / 2;
param.Phase = 0;

cm.RF( param );                % execute instantaneous RF pulse
```

Additional parameters may be supplied to `param`, if partial derivatives with respect to flip angle and/or phase are required or in case of magnetization transfer. See the comments and implementation in `CoMoTk.m` for more details

2.2 Time interval

Executing a time interval of duration `tau`, zero-order gradient moment `p`, and identified by some unique integer index `lambda`, works like this:

```
param = [];
param.lambda = 1;              % this field is always required
param.tau = 0.5;               % required only in first call
param.p = [ 0.1, -0.8, 0.5 ];  % optional, specification in first call is sufficient

cm.time( param );              % execute time interval
```

The field `param.p` is optional, unless the diffusion constant/tensor is nonzero.

Depending on the context, it may be necessary, to add further parameters to `param` (e.g. more details about the gradient shape or to specify bulk motion). For details, look at the examples and the code (which also contains some informative comments).

2.3 Spoiler

An ideal spoiler is a zero duration event, which just eliminates any transverse magnetization. This can be useful to simulate a simplified sequence behaviour (like ideal RF spoiling). It does not require any parameters and is simply invoked as follows:

```
cm.spoiler;                    % execute ideal instantaneous spoiler
```

3 Get results

The CM is applicable to arbitrary sequences and tissues. Interpretation of the results of therefore depends very much on the assumptions of the simulation. To become familiar with the possible approaches, it is therefore crucial to study the CM manuscript and the provided example scripts (which were actually used to generate the figures in the manuscript).

Here, we can only sketch the general strategy:

3.1 Select relevant magnetization pathways ...

The reconstructed signal typically corresponds to some weighted sum over all or a subset of stored configurations. Specific subsets can be extracted with a separate method

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

The elements of the arrays `n` and `lambda`, specify the selected configuration orders and the associated CM dimensions, respectively.

To realize more complicated AND/OR conditions, the (OR-type) `find` method (which returns a boolean array) can be called multiple times with different arguments and the results can be combined with operators `&` and `|`.

The full set of actually stored configurations is always stored in the boolean array `cm.b_n`:

```
b_n = cm.b_n;
```

3.2 ... and calculate their (weighted) sum

Once the subset has been specified in `b_n`, the weighted sum can then be calculated like this

```
param = [];
param.b_n = b_n;

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

The result is returned separately as complex transverse (`res.xy`) and longitudinal (`res.z`) component.

The structure `param` has more optional fields:

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

`x` = Position $\boldsymbol{x}$

`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`
- `w_n` = 1

In addition, arbitrary effects due to inhomogeneous broadening (e.g. caused by susceptibility variations) can be included by supplying a function handle to the field `cm.inhomogeneous_decay`. See the script `bssfp_susc.m` for an example.

Accordingly, any calculated derivatives with respect to `X` are returned as `res.dm_dX.xy` and `res.dm_dX.z`.