Finding OCT pulses

Files required:

```
opt_exc_pulse_circsim_lp_1.m
opt_exc_pulse_circsim_lp_1_repeat.m
opt_ref_pulse_circsim_lp_1.m
opt_ref_pulse_circsim_lp_1_repeat.m
tuned_probe_lp
calc_rot_axis_arba3
tuned_probe_rx
set_params_tapped.m
calc_rot_axis_tuned_probe_lp
sim_spin_dynamics_asymp_mag3
sensitivityParameter.m
quantize_phase.m

plot_hardRef_tuned_probe_lp
plot_opt_ref_results_tuned_probe_lp
plot_opt_exc_results_tuned_probe_lp
calc_time_domain_echo_circsim

calc_masy_tuned_probe_lp.m
cpmg_van_spin_dynamics_asymp_mag4.m

opt_exc_pulse_circsim_inv_lp_repeat.m
opt_exc_pulse_circsim_inv_lp.m

opt_DoubleExc_pulse_circsim_lp_1_repeat.m
opt_DoubleExc_pulse_circsim_lp_1.m
```

Parameters:

Use the file '**set_params_lp_tapped.m'** to change the transmitter parameters. This should be the only file you will have to change.

B0/B1 distributions: currently the pulse finder assumes a uniform B0 distribution. This is set `set_params_lp_tapped.m` by the parameter sp.del_w. This parameter is the offset from the center of the distribution. B1 is determined through sp.sens.

sp.sens is the sensitivity factor of the coil. This parameter has to be determined for each set of coil transmitters. You can run the `sensitivityparameter.m` to find the correct sp.sens for a given T90 and T180. This will plot SNR as a function of sp.sens. Choose the value that gives the first peak. Notice, however, that T180 found by maximizing the signal of a CPMG sequence is not correctly calibrated for finding B1. This parameter will actually be about 1.6*T90, or approximately 140^o. Instead you should calculate T180 by finding the maximum on resonance component of the echo. This can be done by integrating the echo in the time domain, i.e.:

$$S(\omega) = \int_{-\infty}^{\infty} S(t) e^{-i\omega t} dt$$

$$S(0) = \int_{-\infty}^{\infty} S(t) dt$$

The only precaution is to take the integral of the full echo in order to approximate the limits of the integration.

pp.w, sp.w0 and 1/sqrt(sp.L*sp.C) should all be equal so that the carrier frequency and probe are on resonance.

sp.Ron and sp.Roff are determined by matching the rise time and ringdown of a hard pulse with the experimental data. These values will change depending on the Q of the physical transmitter. sp.Ron is the series resistance of the transmitter when it is on, and sp.Roff is the series resistance when the transmitter is off and the Q switch is on. There is a delay incorporated in the code for the time it takes for the Q-switch to turn on. This delay is assumed to be one cycle length.

Procedure:

1. Set Parameters

Begin by setting parameters for the transmitter/receiver in **set_params_lp_tapped.m**

```
sp.fscale = 1;
sp.Ntap = 8; % Autotransformer tap ratio
sp.L = 372.3e-9;
sp.C =  1/(sp.L*(2*pi*0.25*1e6)^2);
sp.R = 7.6e-3;

sp.Ron = 5; %Rs for Tx on .  Adjust these parameters to match rise time and ringdown
sp.Roff = 5; %Rs for Tx off, Q switch on

sp.Rd=960/sp.Ntap^2;  Damping resistance, active damping [Ohms]
sp.Rin=1e6; sp.Cin=10e-12; % Input resistance and capacitance [Ohms, F]
sp.Nrx=4; % Turns ratio of input noise-matching transformer
sp.vn=0.5e-9/sp.Nrx; sp.in=0.1e-12*sp.Nrx; % Resonant receiver input noise voltage [V/sqrt(Hz)] and
current [A/sqrt(Hz)]

% Sample parameters
sp.w0=2*pi*0.25e6*sp.fscale; % Larmor frequency
sp.T=300; % Sample temperature
sp.cycle = 1/(sp.w0/(2*pi));

% Simulation parameters
sp.numpts=2001;  sp.maxoffs=10;
sp.del_w=linspace(-sp.maxoffs,sp.maxoffs,sp.numpts);
sp.sens = 1.95e-4;
sp.mf_type=2; % filter type: 0 -> flat in time, 1 -> matched (white noise), 2 -> matched (colored noise)

% Plotting parameters
sp.plt_axis=0;  sp.plt_tx=0; sp.plt_rx=0;  % 0 to turn these off, 1 to turn plots on


% Pulse sequence parameters
pp.w=2*pi*0.25e6*sp.fscale;
pp.N=16/sp.fscale; % RF frequency, number of quantization steps per cycle
pp.T_90=40e-6; pp.T_180=2*pp.T_90;
pp.NumPhases = 32; %number of phases when discretizing ref pulse
```

2. Optimize Refocusing Pulse

Run **`opt_ref_pulse_circsim_lp_1_repeat.m`** to optimize the refocusing pulse.
This code calls **`opt_ref_pulse_circsim_lp_1.m`** which contains the fit function and the optimization algorithm.

Parameters to set here:

`opt_ref_pulse_circsim_lp_1_repeat(ref_len, techo, file)`

'ref_len': length of the refocusing pulse. This is a required input for the function. Currently the pulse segment is equal to one cycle length. The number of segments is calculated from the length of the pulse, ref_len, and the cycle length. Typically for refocusing pulses, use nref of 1 (mean 1 times T180). SNR, however, is often improved by going to 1.5 or 2 times T180.

'techo': also required, is the echo spacing. The code takes this parameter and calculates the free precession time, params.tfp, which is equal to (techo – t180)/2. Params.tfp is required for the refocusing axis calculation. techo should be in the form N*pi, where pi will be normalized to 2T90.

'file': **`opt_ref_pulse_circsim_lp_1_repeat.m`** requires a file name as an input. This will be where the results of the optimization are saved. The output will be written to this file in 6 columns:

[times   phases   amplitudes   SNR   sp   pp]

The rows are for different optimized pulses with random initial conditions. The times are normalized to T_180, where T_180 = pi. So if the segment time is T_180/10, it is 0.314. The amplitudes are either 1 for the pulse being 'on' or 0 for 'off.'

The SNR value is an absolute value here. In order to find the SNR relative to hard pulses, run **`plot_hardRef_tuned_probe_lp.m`** to find the absolute SNR for hard pulses. This function a set of variables: **`[masy,SNR] = plot_hardRef_tuned_probe_lp(vars)`**
There are four parameters you should include in vars:
'vars.sens': the sensitivity factor that corresponds to the correct B1. The same as sp.sens in **`set_params_lp_tapped`**.

'vars.len': a scaling parameter, which multiplies T90 and T180 from **`set_params_lp_tapped`**. Leave this as 1 to simulate the hard pulses found from experiment and entered in the set params file.

'vars.rat': the ratio of T180/T90 found in from experiment. If T180 is 60us and T90 is 40us, this parameter should be set to 1.5

'vars.techo': This is the same as techo used in refocusing pulse optimization.

Use the ratio SNR_oct/SNR_hardPulses to find the improvement of OCT pulses over hard pulses. Note that the SNR calculation uses hard pulses for both the excitation and refocusing pulses. The SNR calculation for the OCT refocusing pulses assumes the excitation pulse is perfectly matched to the refocusing axis produced by the OCT refocusing pulse.

The code is set up in blocks: the numerical optimization algorithm, the calculation of the coil current using the transmitter model, the spin dynamics calculation, and the filtering through the receiver model. For the refocusing pulse the code works as follows:

1. In **`opt_ref_circsim_lp_1_repeat`**, start with the initial pulse parameters: phases (pref), segment times (tref), and segment amplitudes (aref). The free precession time , params.tfp, is included in these parameters:

   pref = [0 p1 p2 … pn 0]
   tref = [tfp $t_{segment}$ … $t_{segment}$ tfp]
   aref = [0 1 … 1 0]

   We also define the series resistance of the transmitter with the parameter Rsref. Rsref is defined by Rs in the main refocusing pulse finding code, and Rs in turn is determined by Ron and Roff in **`set_params_tapped`**. This definition is:

   Rs = [sp.Ron 1e6 sp.Roff]

   The first value in Rs is the series resistance when the transmitter is on, the second value is the resistance when the transmitter is off but the Q-switch has not turned on yet, and the final value is the resistance with the transmitter off and the Q-switch on.

2. In **`opt_ref_circsim_lp_1`** modify the pulse parameters to account for the time it takes to turn the Q-switch on, and define the series resistance for each time segment:

   ```
   pp.tref=[tfp tref qs (tfp-tqs)]*T_90/(pi/2);
   pp.pref=[0 pref 0 0];
   pp.aref=[0 aref 0 0];
   pp.Rsref=[Rs(1) Rs(3)*ones(1,length(tref)) Rs(2) Rs(1)];
   ```

   Here tqs is the delay for the time between the transmitter turning of and the Q-switch turning on. The factor T_90/(pi/2) converts the normalized units to real time.

3. Calculate the current in the coil using the transmitter model. This is done by the function **`tuned_probe_lp.`** This function uses analytical solutions for the transmitter model to find the coil current. The calculations in this function are currently performed in half-cycle time increments. The code returns a time vector, tvect, and a current vector, Icr. Icr is the current in the rotating frame.

4. The next step is to use the current to calculate the refocusing axis as a function of the offset frequency. This is accomplished by **`calc_rot_axis_arba3`**. The effective rotation axis found by this code is then convolved with the matched filter.

5. Finally the SNR is calculated after passing the rotation axis through the receiver model using **`tuned_probe_rx.`** This function also uses an analytical model to calculate the receiver waveform. The parameter sp.Rd in set params is the series resistance of the receiver. This parameter is current set for active damping. You can vary this to change the receiver Q. The other parameter you can change is the filter type (also in set params):

   sp.mf_type = 0 : rectangular window
   sp.mf_type = 1 : matched filter for white noise
   sp.mf_type = 2 : matched filter for colored noise

   This code integrates the signal over the frequency window and also computes the noise figure. The noise depends largely on the antenna resistance, sp.R. This function also returns the magnetization.

6. The SNR is used as the fit function for the numerical algorithm in **`opt_ref_circsim_lp_1.`** The algorithm, the matlab function fmincon, performs a gradient ascent search to minimize -SNR and then updates the phase parameters. You can change 'lb' and 'ub' in **`opt_ref_circsim_lp_1`** to set the lower and upper bounds for the phase, current these bounds are -2π to 2π. You can also adjust the tolerance of the search in the 'options' parameter. A smaller tolerance will cause the search to continue for a longer time until the SNR reaches a minimum within that tolerance.

7. The main file **`opt_ref_circsin_lp_1_repeat`** then repeats this process many times (defined by 'count') with different initial guesses for the phases.

3. Pick "best" refocusing pulse

Of the solutions found with different initial conditions, you can pick the best to use to optimize the excitation pulse. The simplest choice is the pulse that produces the highest SNR. There are other criteria one might want to consider as well though, such as the echo shape or the smoothness of the pulse. It might be preferable to choose a pulse that has a smooth pulse profile over one that has large jumps in the phase.

Use `plot_opt_ref_results_tuned_probe_lp.m` to plot the current in the coil during the refocusing pulse and refocusing axis:
`[n,SNR] = plot_opt_ref_results_tuned_probe_lp(file,pulse_num)`

'file' is the name of the file for the refocusing pulse.
'pulse_num' is the number of the pulse you want to use in the results cell.
'n' is the magnetization in x, y, and z.

4. Optimize excitation pulse

Now optimize the refocusing pulse. The excitation pulse finding code is `opt_exc_pulse_circsim_lp_1_repeat.m` . This function is structured as
`[SNR] = opt_exc_pulse_circsim_lp_1_repeat(filename,refpulse_num,n)`
This function calls `opt_exc_pulse_circsim_lp_1.m`, which contains the fit function and numerical optimization algorithm.

'refpulse_num' is the number of the pulse from the refocusing pulse file that you want to use.

'lengthExc' is the number of time segments in the excitation pulse, and is in terms of multiples of T180 (actually 2*T90) and should have the form N*pi (lengthExc =2*pi will give an excitation pulse of length 4*T90). This number will be rounded to the nearest cycle length in the code. In order to have excitation pulses that excite a wide enough bandwidth to match the refocusing axis the excitation pulse must be on the order of the echo spacing. If the echo spacing is 7T180, we usually can find good excitation pulses that have a length of 8T180. The number of segments does matter for the excitation pulse. If there are two few pulse segments, the excitation pulse will not be able to excite the higher frequency components. However, too many pulse segments will significantly slow down the optimization. Pulse segments of length 0.1t_180 are a reasonable first attempt. You may want to try a couple different values of nexc and see what gives the best results.

The code is similar to the refocusing pulse in that it consists of several blocks for the transmitter model, spin dynamics, and receiver model. However the calculations are slightly different since we start with the chosen refocusing pulse and no longer assume a perfect excitation pulse. Here are the steps in the excitation pulse finding code:

1. The code `opt_exc_pulse_circsim_lp_1_repeat` starts by loading the parameters for the refocusing pulse. From the refocusing pulse file, we define not only the refocusing pulse itself, but also the free precession time, and the spin parameters. Again this code calculates the refocusing axis, called here 'neff'.
   We also load the excitation pulse parameters, texc, pexc, and aexc. Here we do not need to append the free precession delays to the pulse vectors, as these times are accounted for by the refocusing pulse. We do, however, include the Q-switch delay, 'tqs', and a delay, 'trd', to allow for ringdown of the excitation pulse.

2. With these initial conditions, this code calls `opt_exc_pulse_circsim_lp_1` to perform the numerical optimization. Again this code defines upper and lower bounds ('ub' and 'lb') for the phase, and converts the time vector to real time units and appends the delay segments to the pulse parameters.

3. Again the fit function calculates the SNR. The first step is to calculate the current in the coil do to the excitation pulse using `tuned_probe_lp.` We then have the rotating frame current, 'Icr'. We then remove the timing delays for the ringdown, so that the free precession time starts immediately after the pulse. From the current, the phase and amplitude of the signal are calculated after having being processed through the transmitter model.

4. Now we calculate the spin dynamics using `sim_spin_dynamics_asymp_mag3`. This function uses the excitation pulse parameters as calculated in #3, and the refocusing axis 'neff' to return the asymptotic magnetization for the CPMG sequence, 'masy'.

5. Finally 'masy' is passed into the receiver model using `tuned_probe_rx`. This function again gives the SNR, the negative of which is used as the fit function for fmincon.

6. The main code repeats this process a number of times, defined by the count loop. This optimization will take longer than the refocusing pulse optimization as there are more segments and thus a larger parameter space to optimize over.

7. Note: in addition to the length of the pulse, you may also change the length of the segments. They are currently set to be equal to one cycle length, but you may want to modify 'lexc' in `opt_exc_pulse_circsim_lp_1_repeat` so that the segments are twice or half the cycle length.

5. Pick best excitation pulse

Excitation pulses tend not to have smooth solutions or obvious trends. You can pick the pulse with the highest SNR, but also look at the echo shapes to make sure the echo is symmetric and has a large real component.

6. Plotting the pulses

There are two codes for plotting the echoes produced by either the refocusing pulses or the excitation pulses:

```
plot_opt_ref_results_tuned_probe_lp.m
plot_opt_exc_results_tuned_probe_lp.m
```

Additionally you can plot the pulse produced by hard pulses using:
`plot_hardRef_tuned_probe_lp.m`
The hard pulse plotting function takes the sensitivity parameter as input and outputs the SNR and the asymptotic magnetization: `[masy,SNR] = plot_hardRef_tuned_probe_lp(sens)`

You can use this function to determine the correct sensitivity parameter as mentioned in **#1**. To find the SNR to compare to OCT pulses, use the same value as sp.sens in set_params_lp_tapped. You can turn the plots on in set_params_lp_tapped, or to plot the time domain echo, use

```
calc_time_domain_echo_circsim.m
[echo,tvect]=calc_time_domain_echo_circsim(spect,wvect,plt)
```

7. Determining the Correct Timing

Since the pulses are adjusted to even cycle lengths, it is helpful to calculate the length of the pulses and the correct echo spacing from the results files. For the refocusing pulse, use the first column in the results cell to calculate the length to pulse. This column is a vector:

[Tfp segment1 …. segmentN Tfp]

Where Tfp is the free precession times. The echo spacing is given by the sum of the vector

$$T_{echo} = 2T_{FP} + \sum_{i=1}^{N} segment_i$$

The total length of the refocusing pulse is

$$T_{ref} = \sum_{i=1}^{N} segment_i$$

Where the segment length is equal to one cycle time.

8. Phase Cycling

Phase cycling the OCT pulses is not as simple as adding a pi rotation to the whole pulse. In fact we also have to optimize an excitation pulse that does the proper inversion for phase cycling. Use **`opt_exc_pulse_circsim_inv_lp_repeat.m`** to find phase cycled pulses. This function calls **`opt_exc_pulse_circsim_inv_lp.m`** which optimizes inversion of the input magnetization calculated from the original excitation pulse. This function requires two inputs:

```
opt_exc_pulse_circsim_inv_lp_repeat(file,pulse_num)
```

'file' is the name of the file for the excitation pulse you want to invert.
'pulse_nums' is then number of the pulse within that file.

9. 2-Pulse Excitation

In order to reduce the length of the excitation pulse, it is possible to replace the long pulse by two shorter pulses separated by a delay. The length of the delay, and the length of the two pulses can be varied to try to find a solution that satisfies the constraints of the transmitter.

`opt_DoubleExc_pulse_circsim_lp_1_repeat.m` finds two pulse excitations.
`opt_DoubleExc_pulse_circsim_lp_1_repeat(fileIn,refpulse_num,delay, nexc)`

Again ‘fileIn’ and ‘refpulse_num’ refer to the refocusing pulse to which you want to match the excitation.
‘delay’ is the delay between the pulses. The delay is normalized to T180. delay=2 indicates a delay of 2T180.
‘nexc’ is the number of segments in each pulse (note: the code is currently written for two pulses of equal length).