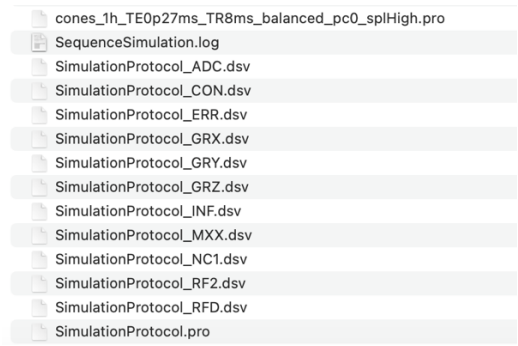
Protocol for Implementing GIRF to 3D Cone Trajectory

This section shows the detailed procedure for implementing GIRF to the simulated gradient waveform on MATLAB®. The algorithm is based on David Leitão and Daniel West's works on EPI trajectory.

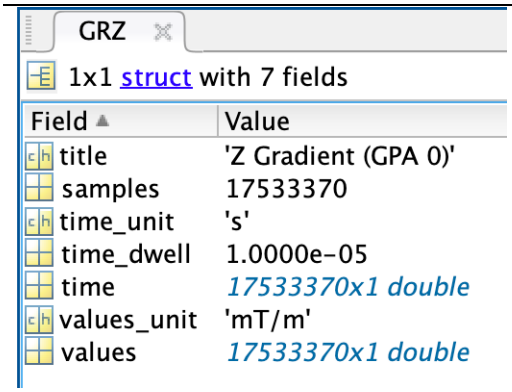


(a)

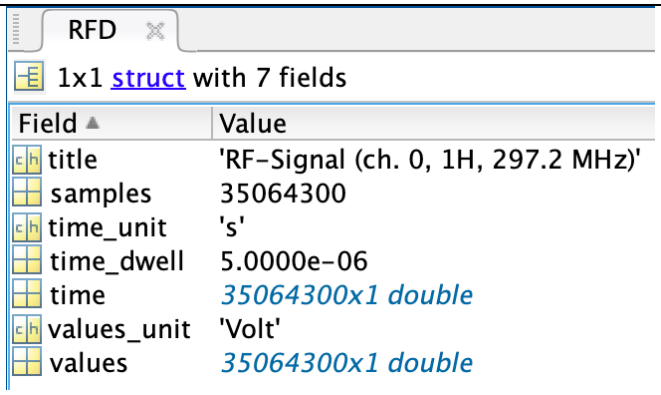
```
%% Load dsv files

protocol = 'SimulationProtocol'; %3D Cone simulation

GRX_file = [protocol,'_GRX.dsv'];
GRY_file = [protocol,'_GRY.dsv'];
GRZ_file = [protocol,'_GRZ.dsv'];
ADC_file = [protocol,'_ADC.dsv'];
RFD_file = [protocol,'_RFD.dsv'];

disp('Reading GRX:');
GRX = dsvread(GRX_file);
disp('Reading GRY:');
GRY = dsvread(GRY_file);
disp('Reading GRZ:');
GRZ = dsvread(GRZ_file);
disp('Reading ADC:');
ADC = dsvread(ADC_file);
disp('Reading RFD:');
RFD = dsvread(RFD_file);
```

(b)

Step 1:Image (a) shows the files obtained after simulation of 3D Cone gradient sequence. Files contain the gradient waveform and RF excitation pulse are ***GRX.dsv***, ***GRY.dsv***, ***GRZ.dsv***, and ***RFD.dsv*** files. Please note that sometimes the RF excitation pulse waveform is stored in ***RF2.dsv***. It would be great to have a check on the ***ADC.dsv*** file to give a brief picture. However, the ADC dwell time in the simulation does not match the actual ADC dwell the scanner use. Therefore, the program itself does not require any information stored in ***ADC.dsv***. The command showed in (b) load the ***.dsv*** file into ***.mat*** files.

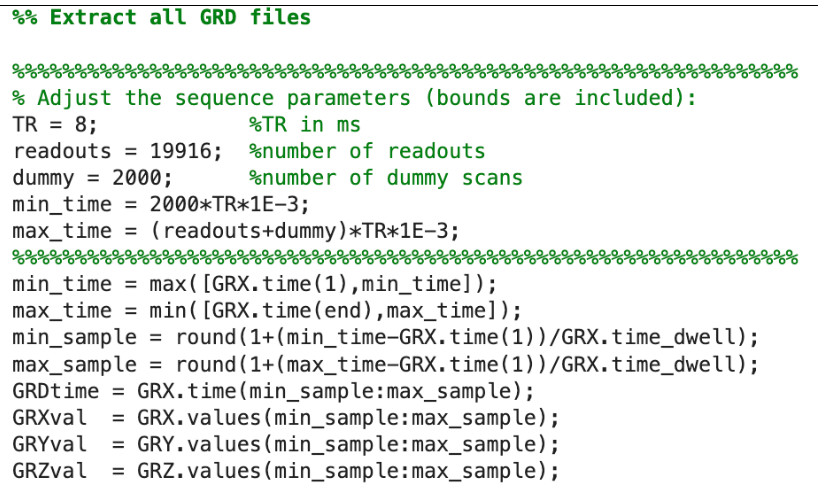


(a)



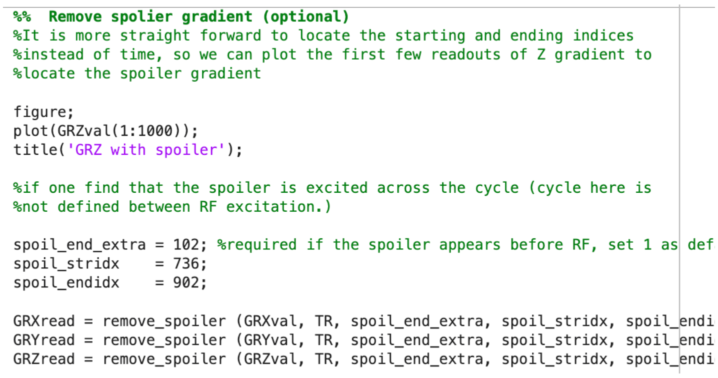
(b)

Step 2: Image (a) shows the variables stored in ***GRZ.dsv*** while image (b) shows the variable stored in ***RFD.dsv*** file. Please note that there is an inconsistency in the dwell time (sampling period) between gradient files and RF excitation waveform file. Our ADC dwell time will be 5us, so that the gradient waveforms have to be interpolated. This will be done in later steps.



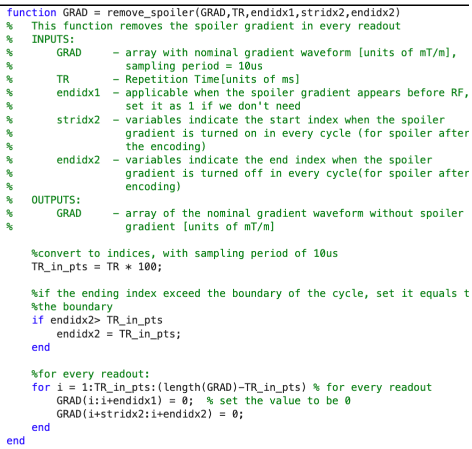
```
%% Extract all GRD files

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adjust the sequence parameters (bounds are included):
TR = 8;             %TR in ms
readouts = 19916;   %number of readouts
dummy = 2000;       %number of dummy scans
min_time = 2000*TR*1E-3;
max_time = (readouts+dummy)*TR*1E-3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
min_time = max([GRX.time(1),min_time]);
max_time = min([GRX.time(end),max_time]);
min_sample = round(1+(min_time-GRX.time(1))/GRX.time_dwell);
max_sample = round(1+(max_time-GRX.time(1))/GRX.time_dwell);
GRDtime = GRX.time(min_sample:max_sample);
GRXval  = GRX.values(min_sample:max_sample);
GRYval  = GRY.values(min_sample:max_sample);
GRZval  = GRZ.values(min_sample:max_sample);
```

Step 3: This shows the command of extracting gradient waveforms based on number of readouts (number of cones), and number of dummy scans. In the bSSFP sequence for proton phantom scans, 2000 dummy scans are programmed to provide better stability of coils.



```
%%  Remove spolier gradient (optional)
%It is more straight forward to locate the starting and ending indices
%instead of time, so we can plot the first few readouts of Z gradient to
%locate the spoiler gradient

figure;
plot(GRZval(1:1000));
title('GRZ with spoiler');

%if one find that the spoiler is excited across the cycle (cycle here is
%not defined between RF excitation.)

spoil_end_extra = 102; %required if the spoiler appears before RF, set 1 as def
spoil_stridx    = 736;
spoil_endidx    = 902;

GRXread = remove_spoiler (GRXval, TR, spoil_end_extra, spoil_stridx, spoil_endi
GRYread = remove_spoiler (GRYval, TR, spoil_end_extra, spoil_stridx, spoil_endi
GRZread = remove_spoiler (GRZval, TR, spoil_end_extra, spoil_stridx, spoil_endi
```

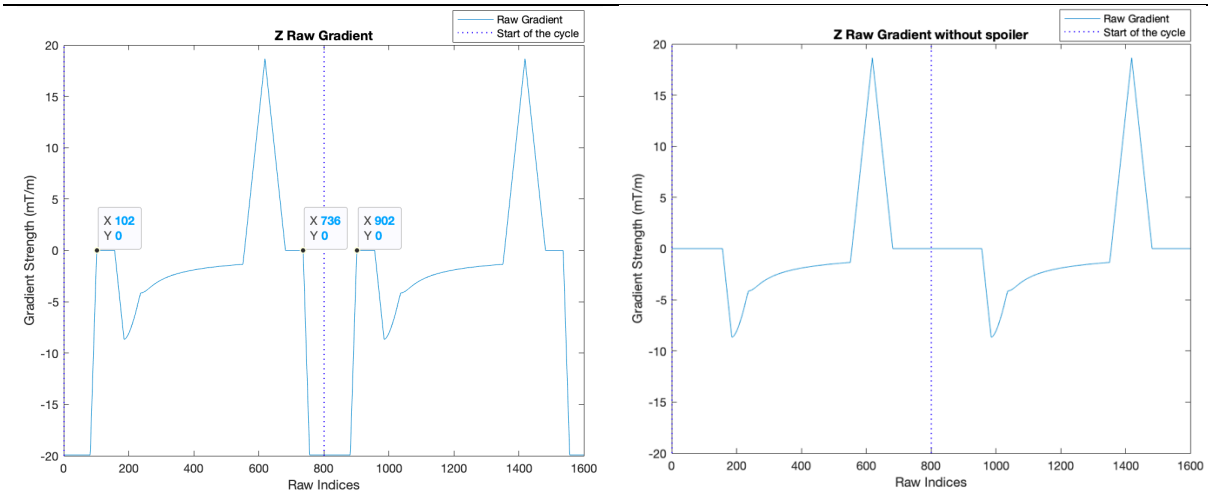
(a)



```
function GRAD = remove_spoiler(GRAD,TR,endidx1,stridx2,endidx2)
%   This function removes the spoiler gradient in every readout
%   INPUTS:
%       GRAD     - array with nominal gradient waveform [units of mT/m],
%                  sampling period = 10us
%       TR       - Repetition Time[units of ms]
%       endidx1  - applicable when the spoiler gradient appears before RF,
%                  set it as 1 if we don't need
%       stridx2  - variables indicate the start index when the spoiler
%                  gradient is turned on in every cycle (for spoiler after
%                  the encoding)
%       endidx2  - variables indicate the end index when the spoiler
%                  gradient is turned off in every cycle(for spoiler after
%                  encoding)
%   OUTPUTS:
%       GRAD     - array of the nominal gradient waveform without spoiler
%                  gradient [units of mT/m]

    %convert to indices, with sampling period of 10us
    TR_in_pts = TR * 100;

    %if the ending index exceed the boundary of the cycle, set it equals t
    %the boundary
    if endidx2> TR_in_pts
        endidx2 = TR_in_pts;
    end

    %for every readout:
    for i = 1:TR_in_pts:(length(GRAD)-TR_in_pts) % for every readout
        GRAD(i:i+endidx1) = 0;  % set the value to be 0
        GRAD(i+stridx2:i+endidx2) = 0;
    end
end
```

(b)

Step 4: After the extraction of raw gradient, the next step will be removing the spoiler gradient in a balanced sequence (if needed) shown in (a). This is done by a function called ***remove_spoiler()*** as shown in (b). The indices of spoiler gradient would be changing according to the pulse sequence. Therefore, it would be more efficient to visually inspect the indices of the start and the end of spoiler gradient by plotted the raw waveform. By finding indices in the first cycle, the function will iteratively remove the remaining spoilers. However. The definition of cycle is different from how we define TR. Detail explanation will be shown in the next step.



(a) (b)

Step 5: The raw Z gradient in (a) indicate the indices to be input in the function. Figure (b) shows the gradient with spoiler gradient removed. Please note that normally the spoiler gradient is programmed after the ramping of the gradient. However, for some sequences, the spoiler gradient is programmed across the cycle due to the short TR. So that part of the spoiler is programmed at the next cycle. The cycle here is not counted between RF excitations. The counting method is shown in (a) by the dotted line. For sequence with longer TR, the variable ***spoil_end_extra*** can be set as 1.

```
%% Apply GIRF
% At this stage we can already apply GIRF to the gradients in the 3
% physical axis GRX, GRY GRZ. For that we'll use the dsv GRD dwell time
% (usually 10us)

load('GIRF_Terra.mat')

[GRX2GRXpre,GRX2B0pre,GRX2GRYpre,GRX2GRZpre] = apply_GIRF_DL(GRXread,GRDtime,GIRF.freq,GIRF.GIRF1xx(:),GIRF.GIRF0x(:),GIRF.GIRF1xy(:),GIRF.GIRI
[GRY2GRYpre,GRY2B0pre,GRY2GRXpre,GRY2GRZpre] = apply_GIRF_DL(GRYread,GRDtime,GIRF.freq,GIRF.GIRF1yy(:),GIRF.GIRF0y(:),GIRF.GIRF1yx(:),GIRF.GIRI
[GRZ2GRZpre,GRZ2B0pre,GRZ2GRXpre,GRZ2GRYpre] = apply_GIRF_DL(GRZread,GRDtime,GIRF.freq,GIRF.GIRF1zz(:),GIRF.GIRF0z(:),GIRF.GIRF1zx(:),GIRF.GIRI

GRXpre = GRX2GRXpre + GRY2GRXpre + GRZ2GRXpre; %[mT/m]
GRYpre = GRX2GRYpre + GRY2GRYpre + GRZ2GRYpre; %[mT/m]
GRZpre = GRX2GRZpre + GRY2GRZpre + GRZ2GRZpre; %[mT/m]
B0pre = GRX2B0pre + GRY2B0pre + GRZ2B0pre; %[mT]
```

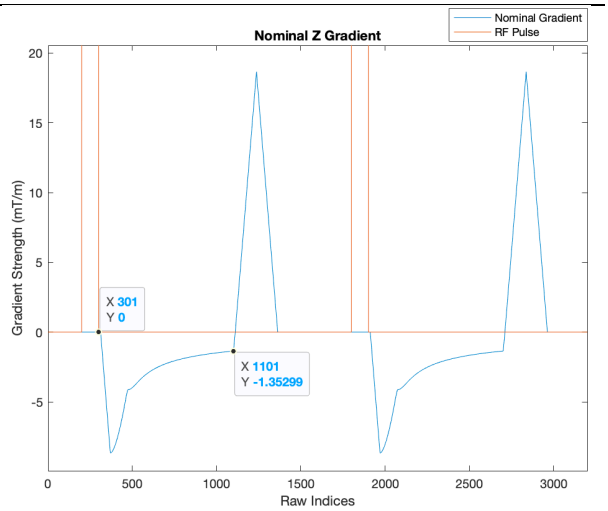
Step 6: This shows the command of applying the GIRF profile to the nominal gradient according to the data and function provided by David Leitão and Daniel West.

```
%% Interpolate the gradient to the ADC dwell time of 5us sampling period

% create a new ADC array with sampling period 5us, which matches the
% sampling period of ADC
ADC_dwell = 5E-6;
newADC = (min_time:ADC_dwell:max_time).';

%Interpolate nominal gradients
GRXnom_interp = interp1(GRDtime,GRXread,newADC);
GRYnom_interp = interp1(GRDtime,GRYread,newADC);
GRZnom_interp = interp1(GRDtime,GRZread,newADC);

%Interpolate GIRF gradients
GRXpre_interp = interp1(GRDtime,GRXpre,newADC);
GRYpre_interp = interp1(GRDtime,GRYpre,newADC);
GRZpre_interp = interp1(GRDtime,GRZpre,newADC);
```

Step 7: This shows the command for interpolating the nominal gradient and GIRF-predicted gradient to the ADC dwell time.

```
%% Calculate the Nominal Trajectory

% Find the index after RF excitation
RF_amp = max(RFD.values);
for i = 2:length(RFD.values)
    if RFD.values(i-1) == RF_amp && RFD.values(i) == 0
        stridx = i;
        break;
    end
end

%It makes the story complicated if we try to find the ending index of the
%trajectory automactically. Let's just plot the gradient and located the
%last index before rewinders

figure;
plot(GRZnom_interp(1:10000)); hold on;
plot(GRZpre_interp(1:10000));
legend('nominal', 'GIRF-predicted')
%find the ending index of the the gradient, e.g. 5ms TE -> 2047, 0.27ms TE
% ->1101
endidx = 1101;
readouts_num = 791; %no. of effective readouts (excluding dead time)

KXnom = traj(GRXnom_interp,stridx,endidx,TR,readouts_num);
KYnom = traj(GRYnom_interp,stridx,endidx,TR,readouts_num);
KZnom = traj(GRZnom_interp,stridx,endidx,TR,readouts_num);
```

(a)

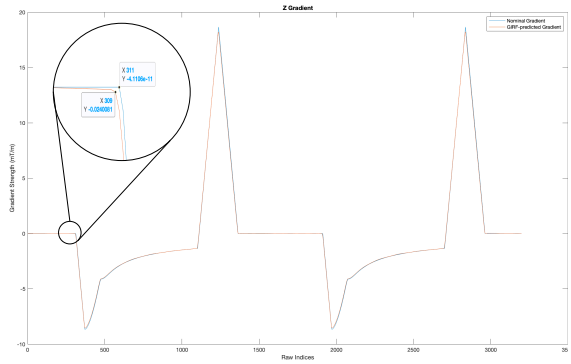


(b)

Step 8: This shows the command for calculating the nominal trajectory in (a). The first index after RF pulse could be searched automatically. However, then ending index of gradient has to be visually inspected in (b). By adjusting ***endidx*** and ***readouts_num*** from different sequence parameters, the trajectory could be obtained by the function ***traj()***.

```
%% Calculate the GIRF-prediceted Trajectory
%If the GIRF-predicted gradients show an apparent delay, the ending index has t
%shift 2 points eariler (i.e. 10us eariler).

figure;
set(gcf,'color','w');
ylabel('Gradient Strength (mT/m)');
xlabel('Raw Indices');
title('Nominal Z Gradient')
plot(GRZnom_interp(1:10000)); hold on;
plot(GRZpre_interp(1:10000));
legend('nominal', 'GIRF-predicted')

endidx_GIRF = endidx - 2;

KXpre = traj(GRXpre_interp,stridx,endidx_GIRF,TR,readouts_num);
KYpre = traj(GRYpre_interp,stridx,endidx_GIRF,TR,readouts_num);
KZpre = traj(GRZpre_interp,stridx,endidx_GIRF,TR,readouts_num);
```

(a)



(b)

Step 9: This shows the command for calculating the GIRF-predicted trajectory in (a). By inspecting the Gradient in (b), apparent delay is corrected by subtracting 2 from ***endidx*** to get the ***endidx_GIRF***. If there is no delay, change 2 to 0 to keep the same ***endidx*** as the nominal one.

```
function result = traj(GRAD,stridx,endidx,TR,readout_pts)
%   This function removes the spoiler gradient in every readout
%   INPUTS:
%       GRAD         - array with gradient waveform after removal of
%                      spoilergradient [units of mT/m]
%       stridx       - variables indicate the index after RF excitation
%       endidx       - variables indicate the end index for time
%                      integration
%       TR           - Repetition Time[units of ms]
%       readout_pts  - variables indicate no. of samples in one readout

%   OUTPUTS:
%       result       - array of the nominal gradient waveform without spoiler
%                      gradient [units of mT/m]

    %no. of samples in one readout, 5us sampling period
    TR_in_pts = TR * 200;
    result = [];
    gamma_HzpmT = 42.577478518e3; % For sodium: 11.26e3;
    ADCdwelltime = 5E-6;
    idx = stridx;

    %no. of samples between RF excitation and end of trajectory (excluding
    %rewinders)
    sample = endidx - idx;

    while idx + sample <= numel(GRAD)
        GRAD_segment = GRAD(idx:idx+sample);
        traj = 2 * pi * gamma_HzpmT *  ADCdwelltime * cumsum(GRAD_segment);
        result = [result; traj(numel(GRAD_segment)-readout_pts+1:end)];
        idx = idx+TR_in_pts;
    end
end
```

Step 10: This shows the function ***traj()*** for calculating the trajectory from the gradient.

```
%% Normalization of Trajectory

%Calculate the absolute maximium of k_max in nominal trajectory
KXmax = double(max(max(KXnom),abs(min(KXnom))));
KYmax = double(max(max(KYnom),abs(min(KYnom))));
KZmax = double(max(max(KZnom),abs(min(KZnom))));

%normalize and concatenate into N*3 matrix, N = no. samples in one readout
%* no. of readouts (no. of Cones). 0.4994,0.4993, and 0.4995 are according
%to the original trajectory from '1mm_traj_dcf_interpolated.mat' in
%Shared Folder
KXnom_nomalized = (KXnom.*0.4994)/KXmax;
KYnom_nomalized = (KYnom.*0.4993)/KYmax;
KZnom_nomalized = (KZnom.*0.4995)/KZmax;
kint_nom = horzcat(KXnom_nomalized,KYnom_nomalized,KZnom_nomalized);

%Apply the same normalization factor from nominal trajectory
KXpre_nomalized = (KXpre.*0.4994)/KXmax;
KYpre_nomalized = (KYpre.*0.4993)/KYmax;
KZpre_nomalized = (KZpre.*0.4995)/KZmax;
kint_pre = horzcat(KXpre_nomalized,KYpre_nomalized,KZpre_nomalized);
```

Step 11: This shows the command to normalize the trajectory, which fits the boundaries of NUFFT operator in the reconstruction algorithm.