Parallel Transmit Challenge Example Code Walkthrough

This challenge is to design the shortest possible multidimensional small-tip-angle parallel transmit excitation RF and gradient pulses, subject to gradient waveform, excitation error, peak RF amplitude and SAR limits. To get you started, we have provided a set of MATLAB scripts that design conventional spokes pulses that satisfy the requirements of the 7T 8-channel torso design scenario, and then evaluate and score those pulses.

`pTxExample.m`
This is the main script, which is intended to be invoked directly from the MATLAB command line. Let's dive into it!

The first command in the script (line 24) loads the evaluation parameter structure (including virtual observation points for SAR calculation), and the B1+ maps and tissue mask. For example, the structure contains the maximum gradient amplitude and slew rate and the maximum RF amplitude, along with other parameters that will be familiar to pulse designers. It also contains some scenario-specific switches for the evaluation code. The **`paralleltxparams_torso.m`** script defines most of these parameters, which is included with the example code package for your benefit but is not used by the example.

```
17      % Load source b1 maps and tissue mask ('maps' structure)
18      % maps.b1: [Nx Ny Nz Nc] (uT), complex B1+ maps
19      % maps.mask: [Nx Ny Nz] (logical), tissue mask
20      % maps.fov: [3] (cm), fov in each dimension
21      %
22      % This will also load the evalp structure defining the problem
23      % constraints
24 -    load('torso_maps.mat');
```

Once the maps and parameters are loaded, we can design a set of spokes pulses for the target slice. The **`dzSpokes`** function for spokes pulse design will have to know several things about the problem, which are picked out of the **`evalp`** structure and used to fill a **`probp`** (**`prob`**lem **`p`**arameters) structure on lines 31-41:

```
27      %% design a spokes pulse
28 -    disp 'Designing a spokes pulse...'
29
30      % Get scalar pulse and constraint parameters
31 -    probp.Nspokes = 5; % number of spokes in pulse
32 -    probp.flipAngle = evalp.flipAngle; % degrees
33 -    probp.beta = 10^-1; % RF power regularization parameter
34 -    probp.slthick = evalp.slThick; % slice thickness, mm
35 -    probp.tb = evalp.tb; % slice profile time-bandwidth product
36 -    probp.dt = 4e-6; % s, dwell time
37 -    probp.maxg = evalp.maxg; % max gradient amplitude, mT/m
38 -    probp.maxgslew = evalp.maxgslew; % max gradient slew, mT/m/ms
39 -    probp.deltaxmin = 10; % finest possible res of kx-ky spokes traj (cm)
40 -    probp.maxb1 = evalp.maxb1; % peak digital RF amplitude constraint
41 -    probp.TR = evalp.TR; % TR for SAR calculation
```

The spokes pulse design only needs the B1+ maps in the center of the target slice, so lines 44-49 pull out the mask and B1+ maps there, and they are used to populate the **`spokesmaps`** structure. The VOP's are copied as well so that the pulse can be dilated in time to meet SAR constraints:

```
43      % get the center slice b1+/df0 maps for the design
44 -    slMapInd = round((evalp.slCent + maps.fov(3)/2)/maps.fov(3)*size(maps.b1,3));
45 -    spokesmaps.b1 = squeeze(maps.b1(:,:,slMapInd,:));
46 -    spokesmaps.mask = squeeze(maps.mask(:,:,slMapInd));
47 -    spokesmaps.fov = maps.fov(1:2);
48 -    spokesmaps.xy = reshape(evalp.xyz(:,1:2),[size(maps.mask) 2]); % design with s
49 -    spokesmaps.xy = squeeze(spokesmaps.xy(:,:,1,:));
50 -    spokesmaps.vop = evalp.vop;
51 -    spokesmaps.maxSAR = evalp.maxSAR;
```

Lines 54-58 calculate the spatial phase pattern of the circularly-polarized mode of the coil array, to use as an initial target phase pattern in the spokes design. In this case this helps improve the resulting flip angle inhomogeneity considerably:

```
53      % get a quadrature initial target phase pattern
54 -    tmp = 0;
55 -    for ii = 1:evalp.Nc
56 -        tmp = tmp + exp(1i*(ii-1)/evalp.Nc*2*pi)*spokesmaps.b1(:,:,ii);
57 -    end
58 -    probp.phsinit = angle(tmp);
```

Finally, we are all set to run the spokes design, which is invoked on line 61. Note the outputs, which are exactly the three needed for the pulse evaluation: the RF waveforms **`rf`**, which have dimensions Ntime x Ncoils, the gradient waveforms **`g`**, which have dimensions Ntime x 3, and the scalar dwell time **`dt`**, in seconds:

```
60      % run the design
61 -    [rf,g,dt] = dzSpokes(spokesmaps,probp);
```

Further details on how the dzSpokes function works can be found in the .zip archive at goo.gl/AUJNda. The outputs from **`dzSpokes`** are then passed to **`pTxEval.m`** to validate and score the pulse:

```
64      %% evaluate the spokes pulse
65 -    disp 'Evaluating the spokes pulse...'
66
67 -    evalp.fName = 'spokes_eval';
68 -    evalp.genfig = true;
69 -    [isValid,dur,errorCode] = pTxEval(rf,g,dt,maps,evalp);
70 -    if isValid == true
71 -        fprintf('Spokes pulses passed with duration %d us\n',dur);
72 -    else
73 -        fprintf('Spokes pulses failed with error code %d\n',errorCode);
74 -    end
```

Note the inputs and outputs to this function: the RF and gradient waveforms and dwell time (which will be supplied by you!) and the maps and evaluation parameters structures (supplied by us). The output is a boolean **`isValid`** reporting whether or not the pulse met all the constraints, the duration of the pulse **`dur`** in microseconds, and the error code if the pulse did not meet a constraint (**`errorCode`**). Note that **`errorCode = -1`** means that the pulse met all constraints, and that the **`pTxEval`** exits after the first constraint is violated, so even though it only reports one violated constraint, it is possible that there are more that are violated.

`pTxEval.m`

This is the evaluation and scoring script. It is long, but pretty simple in function. Lines 50-86 simply check that the RF and gradient vectors are of valid and matching lengths, that the dwell time is a scalar value, etc. Lines 88-115 check that the RF and gradient waveforms meet the amplitude and slew constraints. One subtle requirement is that the first and last samples of the gradient waveforms must be zero. Note that if any of the constraints up to that point are not met, the function will exit with an error, and no score or profile will be calculated:

```
88      % check that peak RF meets constraint
89 -    if max(abs(rf)) > evalp.maxb1
90 -        isValid = false;
91 -        errCode = 8;
92 -        error 'Error code 8: peak RF is too high'
93 -    end
94
95      % check gradient amplitude
96 -    if max(abs(g)) > evalp.maxg
97 -        isValid = false;
98 -        errCode = 9;
99 -        error 'Error code 9: peak gradient is too high'
100 -   end
101
102     % check gradient slew
103 -   gdiff = diff(g)/(dt*1000); % mT/m/ms
104 -   if max(abs(gdiff)) > evalp.maxgslew
105 -       isValid = false;
106 -       errCode = 10;
107 -       error 'Error code 10: peak gradient slew is too high'
108 -   end
109
110     % check that gradient starts and ends at zero
111 -   if g(1) ~= 0 || g(end) ~= 0
112 -       isValid = false;
113 -       errCode = 11;
114 -       error 'Error code 11: gradient waveform doesn''t start+end at 0'
115 -   end
```

The SAR of the pulse is checked on lines 117-134:

```
117     % check that SAR is not too high
118 -   if isfield(evalp,'vop')
119 -       nVOP = size(evalp.vop,3); % # VOPs
120 -       sarVOP = zeros(nVOP,size(rf,1));
121 -       for ii = 1:size(rf,1)
122 -           rft = rf(ii,:); % all coil's samples for this time point
123 -           for jj = 1:nVOP
124 -               sarVOP(jj,ii) = real(rft(:)'*(evalp.vop(:,:,jj)*rft(:)));
125 -           end
126 -       end
127 -       sarVOP = sum(sarVOP,2) * dt./evalp.TR;
128 -       sarVOPfrac = sarVOP./evalp.maxSAR(:);
129 -       if any(sarVOPfrac > 1) % if the pulse violates any SAR constraint
130 -           isValid = false;
131 -           errCode = 12;
132 -           error 'Error code 12: max SAR violated'
133 -       end
134 -   end
```

Note that each constraint is denoted a VOP, and that each VOP can have a different constraint. In the torso case, the first nVOP-1 matrices correspond to local SAR constraints (20 W/kg), and the last constraint is a volume-averaged SAR constraint (4 W/kg).

If your submission has made it this far, now it's time to calculate its excitation pattern. We use a small-tip-angle calculation for this, based on non-uniform Fast Fourier Transforms. First the excitation k-space trajectory is calculated as the time-reversed integral of the gradient waveforms (line 137). Then the trajectory is used to instantiate a **`Gmri`** object (from Jeff Fessler's Image Reconstruction Toolbox, which is a free download at http://web.eecs.umich.edu/~fessler/code/index.html). We take the Hermitian transpose of that object to convert it from an image reconstruction NUFFT to a pulse design NUFFT (line 145). Then we loop over the transmit coils, and calculate each coil's spatial flip angle pattern (units of degrees), weighted by its B1+ field (lines 150-153):

```
136       % calculate the 3D excitation pattern
137 -     k = -flipud(cumsum(flipud(g)))/10*dt*evalp.gamma*100; % exc k-space traj (rad/cm)
138       %t = ((0:size(g,1)-1) - size(g,1))*dt; % time vector for off-resonance
139 -     if evalp.useNUFFT
140
141 -         nufft_args = {size(maps.mask), [7 7 7], 2*size(maps.mask), ...
142             size(maps.mask)/2, 'table', 2^10, 'minmax:kb'};
143 -         mask = maps.mask;
144 -         fov = maps.fov;
145 -         G = Gmri(k/2/pi,mask,'fov',fov,'nufft',nufft_args)';
146 -         b1 = maps.b1;
147 -         b1 = permute(b1,[4 1 2 3]);
148 -         b1 = b1(:,:).';
149 -         theta = 0; % total excited flip angle pattern
150 -         for ii = 1:evalp.Nc
151 -             fprintf('Calculating excitation pattern for transmit coil %d of %d\n',ii,evalp.Nc);
152 -             theta = theta + b1(mask,ii).*(G*rf(:,ii));
153 -         end
154 -         tmp = zeros(size(mask));
155 -         tmp(mask) = theta;
156 -         theta = tmp*1i*evalp.gamma*dt*180/pi;
157 -         clear b1 mask
158
159 -     else
```

If you can't get the Gmri object working on your machine, you can set **`evalp.useNUFFT = false`** to use a brute force (slower) calculation (lines 161-203). Once the pattern is calculated we check whether it meets the in-slice and out-of-slice constraints on maximum and RMS flip angle error. This is done on lines 240-250, and 261-271, using spatial masks contained in the **`evalp`** structure. All these constraints are in units of degrees.

In-between these checks, we also check the through-slice phase profile, to make sure it is refocused (lines 252-259):

```
252 -       phs = unwrap(angle(theta(:,:,evalp.inSliceInds)),[],3);
253 -       phsd = phs - repmat(mean(phs,3),[1 1 size(phs,3)]);
254 -       if any(abs(phsd(:)) > evalp.maxPhsDev)
255 -           isValid = false;
256 -           errCode = 15;
257 -           errMsg = 'Error code 15: Through-slice phase profile does not meet constraints';
258 -           error(errMsg);
259 -       end
```

This is done by unwrapping the through-slice phase, subtracting the mean phase, and checking whether there are any points in the through-slice dimension that are more than **`evalp.maxPhsDev`** away from the mean through-slice phase in each location.

If your pulse has made it this far, congratulations, it will be scored! That is done on line 274:

```
273         % calculate the pulse duration
274 -       dur = ceil(dt*size(rf,1)*1000000); % milliseconds
275
276         % if they made it this far, the pulse is valid
277 -       isValid = true;
278
279         % save everything in output file, exit
280 -       if isfield(evalp,'fName')
281 -           save(evalp.fName);
282 -       end
```

The duration is calculated in microseconds. Since the pulse has met all constraints at this point, isValid is set to true, and the evaluation's workspace is optionally saved to a .mat file for debugging.

That's it - **Good luck!!!**