mapVBVD - a short Tutorial

Table of Contents

Philipp Ehses philipp.ehses@tuebingen.mpg.de Created: September 2016

Parse twix file

The basic idea of mapVBVD is to divide reading the data into two parts: First, parsing the twix file and saving all relevant information from the 'measurement data headers' (mdh) for each acquired line of k-space together with their position in the file (byte position). Second, read the data into memory after choosing which data should be read (which slice/s, which repetition/s, which line/s, ...). This is similar to memory-mapping, hence the name.

Parsing of the data can be achieved in several ways:

```
% 1) open dialog to choose file from
% twix = mapVBVD();

% 2) use full filename (including path if necessary)
% twix =
 mapVBVD('meas_MID00501_FID24867_t2_tse_tra_192_4mm_2ave.dat');

% 3) use measurement id (you need to 'cd' to the containing folder)
twix = mapVBVD(501);

Software version: VD (!?)
Scan 1/2, read all mdhs:
    100 % read in    0 s; estimated time left:    0 s
          parse mdhs... done. (0.010702 seconds)
Scan 2/2, read all mdhs:
    100 % read in    0 s; estimated time left:    0 s
          parse mdhs... done. (0.012108 seconds)
```

First look at 'twix'

There are two cells in twix (this is a multi-raid VD/VE file).

```
twix
```

```
twix =

    [1x1 struct]    [1x1 struct]
```

The first 'measurement' includes the noise data and its header, the second holds the image data (as well as other data such as phasecor, refscan,...). Note that twix data from VA and VB systems will never show multiple measurements and 'twix' will simply be a struct with the contents of twix{2}.

```
twix{:}


ans =

      hdr: [1x1 struct]
    noise: [1x1 twix_map_obj]


ans =

         hdr: [1x1 struct]
       image: [1x1 twix_map_obj]
    phasecor: [1x1 twix_map_obj]
```

In this tutorial we do not care for noise data, so let's get rid of the first measurement:

```
twix = twix{2};
```

A closer look

Now let's look at the object that holds the image information

```
twix.image


ans =

  twix_map_obj with properties:

               flagRemoveOS: 0
              flagDoAverage: 0
            flagAverageReps: 0
            flagAverageSets: 0
              flagIgnoreSeg: 0
        flagSkipToFirstLine: 0
         flagRampSampRegrid: 0
       flagDoRawDataCorrect: 0
    RawDataCorrectionFactors: []
             flagAverageDim: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
                   filename: '/home/ehses/mapVBVD_tutorial/
meas_MID00501_...'
            softwareVersion: 'vd'
                   dataType: 'image'
```

```
                rampSampTrj: []
                   dataSize: [384 8 198 1 3 2 1 1 1 1 11 1 1 1 1 1]
                    sqzSize: [384 8 198 3 2 11]
                   dataDims: {1x16 cell}
                    sqzDims: {'Col'  'Cha'  'Lin'  'Sli'  'Ave'
'Seg'}
                       NCol: 384
                       NCha: 8
                       NLin: 198
                       NPar: 1
                       NSli: 3
                       NAve: 2
                       NPhs: 1
                       NEco: 1
                       NRep: 1
                       NSet: 1
                       NSeg: 11
                       NIda: 1
                       NIdb: 1
                       NIdc: 1
                       NIdd: 1
                       NIde: 1
                       NAcq: 1188
                        Lin: [1x1188 double]
                        Par: [1x1188 double]
                        Sli: [1x1188 double]
                        Ave: [1x1188 double]
                        Phs: [1x1188 double]
                        Eco: [1x1188 double]
                        Rep: [1x1188 double]
                        Set: [1x1188 double]
                        Seg: [1x1188 double]
                        Ida: [1x1188 double]
                        Idb: [1x1188 double]
                        Idc: [1x1188 double]
                        Idd: [1x1188 double]
                        Ide: [1x1188 double]
                  centerCol: [1x1188 double]
                  centerLin: [1x1188 double]
                  centerPar: [1x1188 double]
                     cutOff: [2x1188 double]
                 coilSelect: [1x1188 double]
                ROoffcenter: [1x1188 double]
                timeSinceRF: [1x1188 double]
                IsReflected: [1x1188 logical]
            IsRawDataCorrect: [1x1188 logical]
                    slicePos: [7x1188 double]
                  freeParam: [4x1188 double]
                    iceParam: [24x1188 double]
                scancounter: [1x1188 double]
                  timestamp: [1x1188 double]
                    pmutime: [1x1188 double]
                     memPos: [1x1188 double]
               isBrokenFile: 0
```

As you can see there is a lot of information, it seems that we acquired three slices, two averages and 11 segments. Here's the matrix size that would be created if we would call twix.image():

```
twix.image.dataSize(1:11)
```

```
ans =

   384     8   198     1     3     2     1     1     1     1    11
```

Making use of flags

Let's get rid of the average dimension by telling the twix object to automatically average them during the read procedure. In addition, we also want to get rid of the oversampling that is almost always performed in read direction (2 x oversampling = 2 x larger FoV in read):

```
twix.image.flagDoAverage = true;
twix.image.flagRemoveOS  = true;
```

Btw, flagDoAverage is a shortcut to flagAverageDim, and in fact we can average every of the 16 dimensions by setting the corresponding entry in flagAverageDim to true;

Now let's see how the two flags changed the 'size' of our data:

```
twix.image.dataSize(1:11)
```

```
ans =

   192     8   198     1     3     1     1     1     1     1    11
```

The average dimension is gone and the 'Col'(read)-dimension is only half its previous size. This will greatly reduce our memory footprint...

There are a bunch of our flags - most of them are just shortcuts to flagAverageDim. 'flagRampSampRegrid' may be of interest for people working with EPI data as it tells the twix-object to automatically regrid rampsampled data during the read procedure.

First look at our data

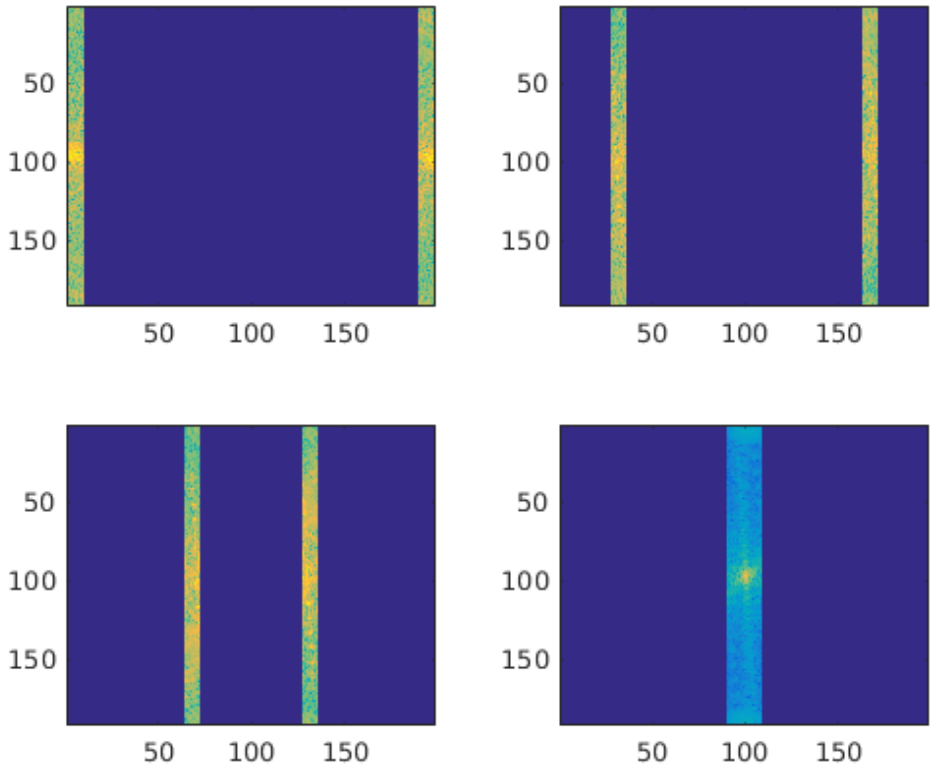
Now, let's look at different segments of our data. What are segments? Segments indicate how the acquisition of the data was 'segmented', e.g. in an EPI or TSE acquisition. This data is from a TSE and the number of segments is equal to the turbo factor. So each of this segments corresponds to a different echo in our TSE echo train.

Let's read in the second slice of our data. There are ways to shorten the following expression, but this is probably the safest way:

```
%                 Col Cha Lin Par Sli Ave Phs Eco Rep Set Seg
data = twix.image(  :,  :,  :,  1,  2,  1,  1,  1,  1,  1, :);
```

Now we look at four segments for one coil:

```
subplot(2,2,1);
imagesc(abs(squeeze(data(:,1,:, 1))).^0.2);
subplot(2,2,2);
imagesc(abs(squeeze(data(:,1,:, 4))).^0.2);
subplot(2,2,3);
imagesc(abs(squeeze(data(:,1,:, 8))).^0.2);
subplot(2,2,4);
imagesc(abs(squeeze(data(:,1,:,11))).^0.2);
```



As you can see, each segment holds a continuous 'segment' of k-space. There are lots of zeros in the data, so our data matrix is much larger than it needs to be. The segment information can be important for phase correction but let's omit it in order to reduce the size of our data:

```
twix.image.flagIgnoreSeg = true;

% Let's look again at the data size to see that the segments dim. is
 gone:
twix.image.dataSize(1:11)


ans =

   192     8   198     1     3     1     1     1     1     1     1
```

By the way, I got quite a few emails from people that had problems with their data not fitting into their computer's memory. This flag as well as the other averaging flag can help a lot!

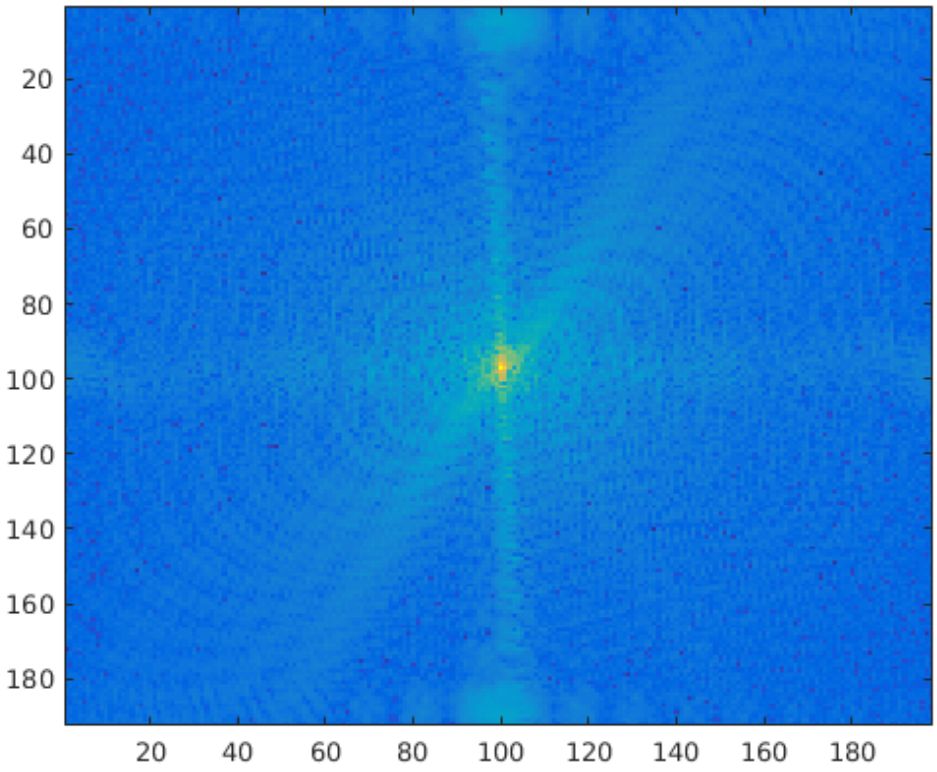
Let's read in the data again. This time we can read in all slices at once, our matrix is much smaller now:

```
data = twix.image();
size(data)
```

```
ans =

   192     8   198     1     3
```

And here's the k-space for the first coil and the first slice:

```
figure,imagesc(abs(squeeze(data(:,1,:,1))).^0.2);
```



Let's reconstruct the data

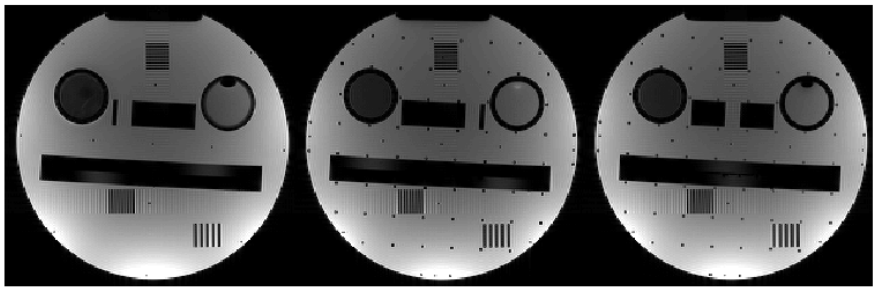
Now, we're ready to reconstruct the data. For large datasets it can be very important to never read in all the data into memory but only read in as much data as is required for each reconstruction step. In this example, we can reconstruct each slice separately in order to reduce our memory footprint.

```
%
```

```
% The final matrix size of our image (single precision is enough):
os  = 2; % oversampling factor
img = zeros([twix.image.NCol/os, twix.image.NLin,
 twix.image.NSli], 'single');

for sli = 1:twix.image.NSli
    % read in the data for slice 'sli'
    data = twix.image(:,:,:,1,sli);

    % fft in col and lin dimension:
    fft_dims = [1 3];
    for f = fft_dims
        data = ifftshift(ifft(fftshift(data,f),[],f),f);
    end

    % sum-of-square coil combination:
    img(:,:,sli) = squeeze(sqrt(sum(abs(data).^2,2)));
end
```

And here's the result for the three slices:

```
scrsz = get(groot,'ScreenSize');
figure('Position',[1 scrsz(4)/4 scrsz(3)/2 scrsz(4)/4])
imagesc(flipud(img(:,:)), [0, 0.7*max(img(:))]), colormap gray,
 axis image off;
```



Appendix: Header information

In case you are looking for header information, the twix-structure also contains a 'hdr' entry. It contains all the header information that was recognized by read_twix_hdr.m. It's a very basic script, so unfortunately some header information is lost.

```
twix.hdr
```

```
ans =

      Config: [1x1 struct]
       Dicom: [1x1 struct]
        Meas: [1x1 struct]
    MeasYaps: [1x1 struct]
```

```
    Phoenix: [1x1 struct]
      Spice: [1x1 struct]
```

There are different types of headers with a lot of redundant data. I usually use the 'MeasYaps' structure which includes most of the stuff that you can access in IDEA via 'MrProt' (also, same structure):

A few examples:

```
TR = twix.hdr.MeasYaps.alTR{1}  %us
TE = twix.hdr.MeasYaps.alTE{1}  %us

sli_thickness = twix.hdr.MeasYaps.sSliceArray.asSlice{1}.dThickness
sli_position  = twix.hdr.MeasYaps.sSliceArray.asSlice{1}.sPosition

turbo_factor = twix.hdr.MeasYaps.sFastImaging.lTurboFactor
```

```
TR =

    2000000


TE =

     100000


sli_thickness =

    4


sli_position =

    dSag: 1.8072
    dCor: -1.2048
    dTra: -14.0241


turbo_factor =

    11
```

Published with MATLAB® R2015b