demo_read_3d_gre

December 26, 2024

1 Siemens scan reading demo

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

from read_siemens_scans.twix_reading import read_raw_twix_data
from read_siemens_scans.recon import adjoint_recon, kspace_fix_aspect_ratio
from read_siemens_scans.visualize import imshow
from read_siemens_scans.config import DIM_DESCRIPTION

%matplotlib inline
```

1.0.1 Configure the directory of TWIX files

[9]:
```
dataset_path = '/mnt/c/Users/along/Downloads/In-vivo twix files'
```

1.0.2 Load file paths, display index of loaded paths

[10]:
```
index_dict = dict()
index_script = Path(dataset_path).joinpath('index.py')
with open(index_script, 'r') as file: exec(file.read(), {"__file__":␣
 ↪index_script}, index_dict)
print('Loaded the following dictionaries:\n')
for key, content in index_dict.items():
    if key in ['Path', '_dir']: continue
    print(f'{key}:\n' + f'\t\tKeys:\t\t' + ", ".join(content.keys()) + '\n')

locals().update(index_dict)
```

```
Loaded the following dictionaries:

old_mrzero_tube:
                Keys:           Baseline MRzero

old_mrzero_human:
                Keys:           Baseline MRzero exp. 1, Baseline MRzero exp. 2
```

```
amir_3d_gre:
                Keys:           Try 1, Unknown, Ball Dec 25

pulseq_2d_ball:
                Keys:           HASTE, TSE
```

1.0.3 Choose a file to read

[11]:
```
# Choose a dictionary (experiment set), and key (scan)
twix_path = amir_3d_gre['Ball Dec 25']
```

1.0.4 Read k-space, reconstruct in raw form

[12]:
```
print(f'Reading file \t {twix_path}\n')
kspace, dim_data = read_raw_twix_data(twix_path.__str__(),␣
 ↪fix_oversampling=False)
images = adjoint_recon(kspace, dim_data)

# NOTE: Showing adjoint reconstruction. Proper recon for HASTE is not supported.
```

```
Reading file     /mnt/c/Users/along/Downloads/In-vivo twix
files/meas_MID00210_FID12025_pulseq.dat

Software version: VD/VE (!?)

Scan  0

100%|      | 118M/118M [00:04<00:00, 27.1MB/s]

Scan  1

100%|      | 224M/224M [00:09<00:00, 23.6MB/s]
```

1.0.5 Print file dimensions info

It is an ordered dictionary of pairs in the format: *"Name of dimension (name of dimension in the file)": length at that dimension.*

self.print() shows its content.

[13]:
```
dim_data.print()
```

```
{
    "Axis 1 - probably phase encoding 1 (Par)": 64,
    "Axis 2 - possibly readout (Lin)": 64,
    "Coil (Cha)": 54,
    "Axis 3 - probably phase encoding 2 (Col)": 128
}
```

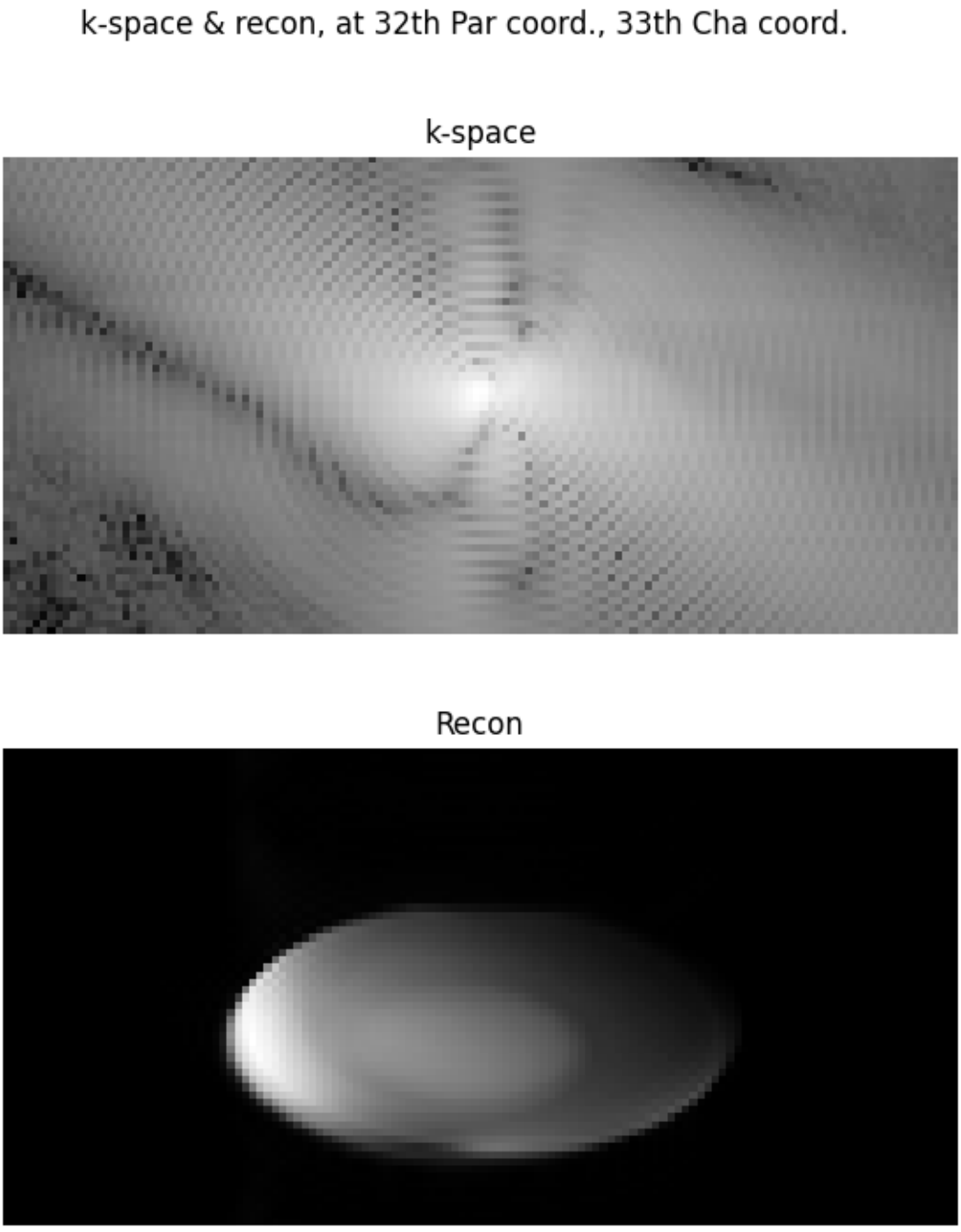
Look carefully at the dimensions printed above.

This notebook is comprised of **two parts**. Choose the right one depending on whether the experiment is 3D (two phase encoding dims) or 2D (one phase encoding).

2 Displaying 3D data

[14]:
```
# Choice of images
par_ind = 32
cha_ind = 33

fig, ax = plt.subplots(2, 1)
fig.suptitle(f'k-space & recon, at {par_ind}th Par coord., {cha_ind}th Cha␣
 ↪coord.')
ax[0].set_title(f'k-space')
imshow(ax[0], 10 * np.log10(abs(kspace[par_ind, :, cha_ind, :])))
ax[1].set_title(f'Recon')
imshow(ax[1], abs(images[par_ind, :, cha_ind, :]))
fig.set_size_inches(7, 8)
```



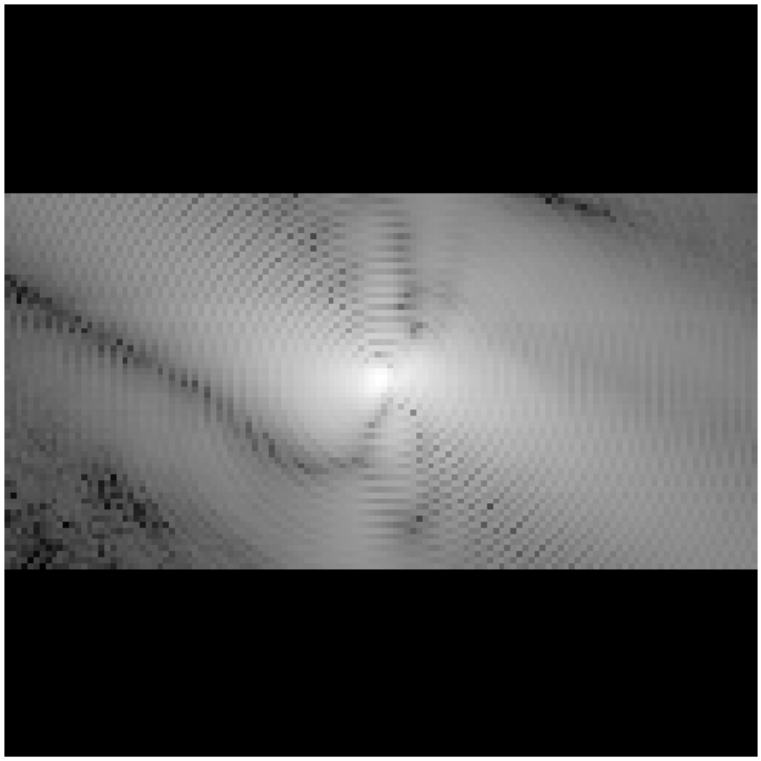
2.0.1 Turn aspect ratio into 1:1 by a simple Fourier interpolation

[15]:
```
kspace_fixed = kspace_fix_aspect_ratio(kspace, dim_data)
images_interp = adjoint_recon(kspace_fixed, dim_data)
```

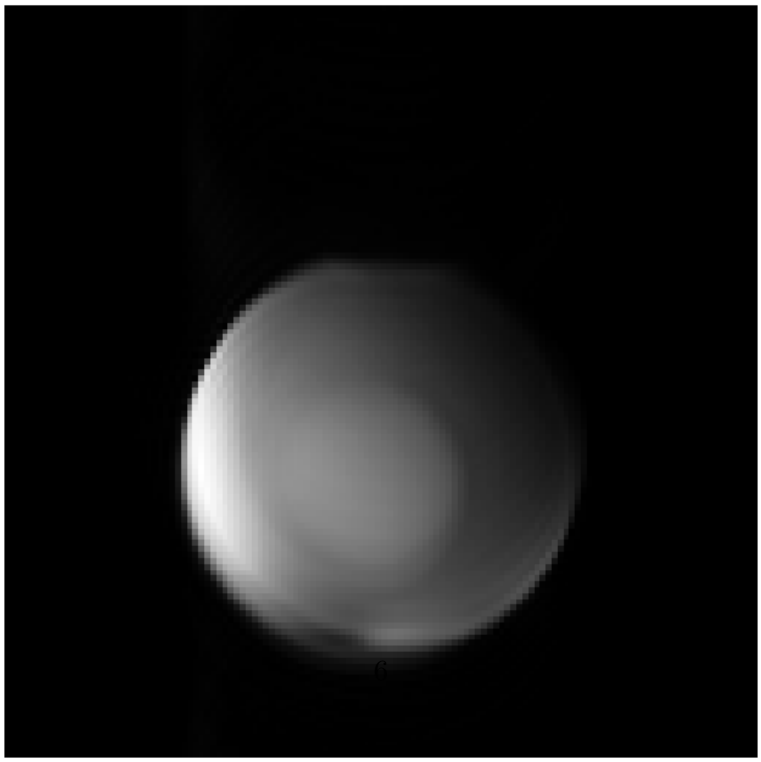
[16]:
```
fig, ax = plt.subplots(2, 1)
fig.suptitle(f'Same k-space & recon, after interpolation')
ax[0].set_title(f'k-space')
kspace_floor = abs(kspace[par_ind, :, cha_ind, :]).min()
imshow(ax[0], 10 * np.log10(np.maximum(abs(kspace_fixed[par_ind, :, cha_ind, :
 ↪]), kspace_floor)))
ax[1].set_title(f'Recon')
imshow(ax[1], abs(images_interp[par_ind, :, cha_ind, :]))
fig.set_size_inches(7, 14)
```

Same k-space & recon, after interpolation

k-space



Recon

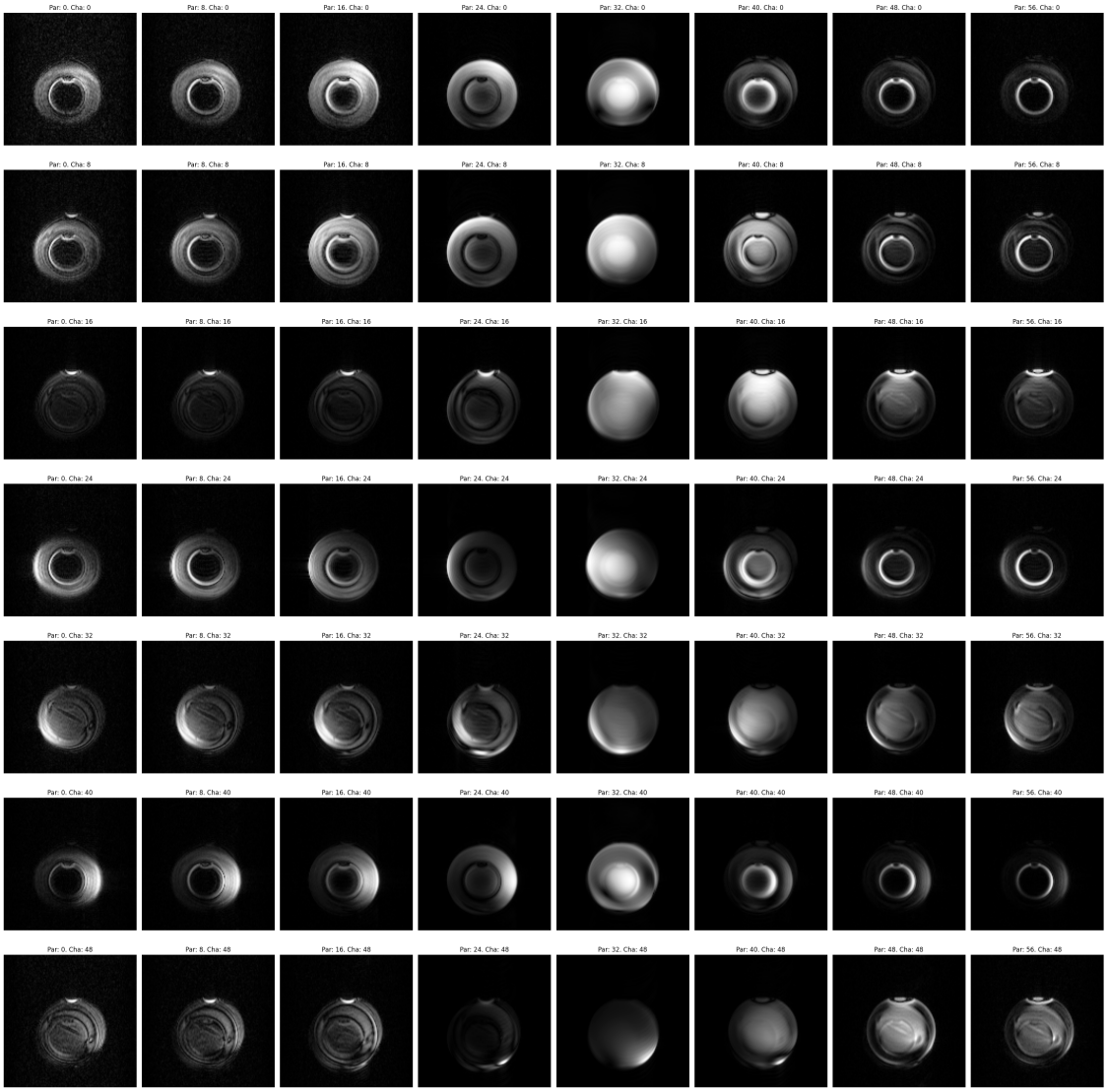


2.0.2 Plotting all such images together

[17]:
```
size_cha = dim_data[DIM_DESCRIPTION['Cha']]
size_par = dim_data[DIM_DESCRIPTION['Par']]
skip = 8

cha_inds = list(range(0, size_cha, skip))
par_inds = list(range(0, size_par, skip))
assert len(cha_inds) > 1
assert len(par_inds) > 1

fig, ax = plt.subplots(len(cha_inds), len(par_inds))
for cha_ind, cha in enumerate(cha_inds):
    for par_ind, par in enumerate(par_inds):
        ax[cha_ind, par_ind].set_title(f'Par: {par}. Cha: {cha}')
        imshow(ax[cha_ind, par_ind], abs(images_interp[par, :, cha, :]))
fig.set_size_inches(30, 30)
fig.tight_layout()
```



3 Displaying 2D data

[ ]:
```
# Choice of images
cha_ind = 2

fig, ax = plt.subplots(2, 1)
fig.suptitle(f'k-space & recon, at {cha_ind}th Cha coord.')
ax[0].set_title(f'k-space')
imshow(ax[0], 10 * np.log10(abs(kspace[:, cha_ind, :])))
ax[1].set_title(f'Recon')
imshow(ax[1], abs(images[:, cha_ind, :]))
fig.set_size_inches(7, 8)
```

3.0.1 Turn aspect ratio into 1:1 by a simple Fourier interpolation

[11]:
```
kspace_fixed = kspace_fix_aspect_ratio(kspace, dim_data)
images_interp = adjoint_recon(kspace_fixed, dim_data)
```

[ ]:
```
fig, ax = plt.subplots(2, 1)
fig.suptitle(f'Same k-space & recon, after interpolation')
ax[0].set_title(f'k-space')
kspace_floor = abs(kspace[:, cha_ind, :]).min()
imshow(ax[0], 10 * np.log10(np.maximum(abs(kspace_fixed[:, cha_ind, :]),␣
 ↪kspace_floor)))
ax[1].set_title(f'Recon')
imshow(ax[1], abs(images_interp[:, cha_ind, :]))
fig.set_size_inches(7, 14)
```

3.0.2 Plotting all such images together

[ ]:
```
size_cha = dim_data[DIM_DESCRIPTION['Cha']]
skip = 1

cha_inds = list(range(0, size_cha, skip))
assert len(cha_inds) > 1

fig, ax = plt.subplots(1, len(cha_inds))
for cha_ind, cha in enumerate(cha_inds):
    ax[cha_ind].set_title(f'Cha: {cha}')
    imshow(ax[cha_ind], abs(images_interp[:, cha, :]))
fig.set_size_inches(30, 7)
fig.tight_layout()
```