Model Based Parameter Quantification Documentation

Release 2020.0.31beta1

Oliver Maier

Oct 04, 2021

Contents

3D model based parameter quantification for MRI.

PyQMRI is a Python module to quantify tissue parameters given a set of MRI measurements, specifically desinged to quantify the parameter of interest. Examples include T1 quantification from variable flip angle or inversion-recovery Look-Locker data, T2 quantification using a mono-exponential fit, or Diffusion Tensor quantification.

In addition, a Genereal Model exists that can be invoced using a text file containing the analytical signal equation.

For a real world usage example have a look at the *Quickstart Guide*. The example can also be run interactively using GoogleColab.

CHAPTER 1

Quickstart Guide

1.1 Installation

First, setup the prerequest for **PyQMRI**:

[ ]:
```
!apt-get update
!apt-get install libclfft-dev -y
```

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu bionic-cran40/ InRelease
Ign:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64 ␣
↪InRelease
Hit:3 http://security.ubuntu.com/ubuntu bionic-security InRelease
Ign:4 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/
↪x86_64  InRelease
Hit:5 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64 ␣
↪Release
Hit:6 https://developer.download.nvidia.com/compute/machine-learning/repos/ubuntu1804/
↪x86_64  Release
Hit:7 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:8 http://ppa.launchpad.net/c2d4u.team/c2d4u4.0+/ubuntu bionic InRelease
Hit:9 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:10 http://archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:11 http://ppa.launchpad.net/graphics-drivers/ppa/ubuntu bionic InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
libclfft-dev is already the newest version (2.12.2-1build2).
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
```

[ ]:
```
!git clone https://github.com/geggo/gpyfft.git
```

```
fatal: destination path 'gpyfft' already exists and is not an empty directory.
```

[ ]:
```
!pip install gpyfft/.
```

```
Processing ./gpyfft
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from␣
↪gpyfft==0.7.3) (1.18.5)
Requirement already satisfied: pyopencl in /usr/local/lib/python3.6/dist-packages␣
↪(from gpyfft==0.7.3) (2020.2.2)
Requirement already satisfied: appdirs>=1.4.0 in /usr/local/lib/python3.6/dist-
↪packages (from pyopencl->gpyfft==0.7.3) (1.4.4)
Requirement already satisfied: decorator>=3.2.0 in /usr/local/lib/python3.6/dist-
↪packages (from pyopencl->gpyfft==0.7.3) (4.4.2)
Requirement already satisfied: pytools>=2017.6 in /usr/local/lib/python3.6/dist-
↪packages (from pyopencl->gpyfft==0.7.3) (2020.4.3)
Requirement already satisfied: dataclasses>=0.7; python_version <= "3.6" in /usr/
↪local/lib/python3.6/dist-packages (from pytools>=2017.6->pyopencl->gpyfft==0.7.3)␣
↪(0.7)
Requirement already satisfied: six>=1.8.0 in /usr/local/lib/python3.6/dist-packages␣
↪(from pytools>=2017.6->pyopencl->gpyfft==0.7.3) (1.15.0)
Building wheels for collected packages: gpyfft
  Building wheel for gpyfft (setup.py) ... done
  Created wheel for gpyfft: filename=gpyfft-0.7.3-cp36-cp36m-linux_x86_64.whl␣
↪size=367757 sha256=609c21b4b3d4d1516f80522ca6085730f8f1b3ac6831e3e50ef0ae94f0c4ec8d
  Stored in directory: /tmp/pip-ephem-wheel-cache-hgbwtlsh/wheels/59/8e/ab/
↪202c7b2221f0b7acf0f9b0f7ce9c9c7697830acdd5bd6d7aa5
Successfully built gpyfft
Installing collected packages: gpyfft
  Found existing installation: gpyfft 0.7.3
    Uninstalling gpyfft-0.7.3:
      Successfully uninstalled gpyfft-0.7.3
Successfully installed gpyfft-0.7.3
```

After the successfull installation of the prerequests, we can install **PyQMRI** itself:

[ ]:
```
!pip install pyqmri
```

```
Requirement already satisfied: pyqmri in /usr/local/lib/python3.6/dist-packages (0.3.
↪2.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages␣
↪(from pyqmri) (3.2.2)
Requirement already satisfied: pyfftw in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (0.12.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (2.10.0)
Requirement already satisfied: ipyparallel in /usr/local/lib/python3.6/dist-packages␣
↪(from pyqmri) (6.3.0)
Requirement already satisfied: cython in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (0.29.21)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (1.18.5)
Requirement already satisfied: numexpr in /usr/local/lib/python3.6/dist-packages␣
↪(from pyqmri) (2.7.1)
Requirement already satisfied: pyqt5 in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (5.15.1)
Requirement already satisfied: mako in /usr/local/lib/python3.6/dist-packages (from␣
↪pyqmri) (1.1.3)
Requirement already satisfied: sympy>=1.6.2 in /usr/local/lib/python3.6/dist-packages␣
↪(from pyqmri) (1.6.2)
Requirement already satisfied: pyopencl in /usr/local/lib/python3.6/dist-packages␣
↪(from pyqmri) (2020.2.2)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages␣
↪(from matplotlib->pyqmri) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-
↪packages (from matplotlib->pyqmri) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/
↪lib/python3.6/dist-packages (from matplotlib->pyqmri) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-
↪packages (from matplotlib->pyqmri) (1.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from␣
↪h5py->pyqmri) (1.15.0)
Requirement already satisfied: traitlets>=4.3 in /usr/local/lib/python3.6/dist-
↪packages (from ipyparallel->pyqmri) (4.3.3)
Requirement already satisfied: ipykernel>=4.4 in /usr/local/lib/python3.6/dist-
↪packages (from ipyparallel->pyqmri) (4.10.1)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.6/dist-
↪packages (from ipyparallel->pyqmri) (5.3.5)
Requirement already satisfied: tornado>=4 in /usr/local/lib/python3.6/dist-packages␣
↪(from ipyparallel->pyqmri) (5.1.1)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/dist-
↪packages (from ipyparallel->pyqmri) (0.2.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.6/dist-packages␣
↪(from ipyparallel->pyqmri) (19.0.2)
Requirement already satisfied: ipython>=4 in /usr/local/lib/python3.6/dist-packages␣
↪(from ipyparallel->pyqmri) (5.5.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-packages␣
↪(from ipyparallel->pyqmri) (4.4.2)
Requirement already satisfied: PyQt5-sip<13,>=12.8 in /usr/local/lib/python3.6/dist-
↪packages (from pyqt5->pyqmri) (12.8.1)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.6/dist-
↪packages (from mako->pyqmri) (1.1.1)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.6/dist-packages␣
↪(from sympy>=1.6.2->pyqmri) (1.1.0)
Requirement already satisfied: pytools>=2017.6 in /usr/local/lib/python3.6/dist-
↪packages (from pyopencl->pyqmri) (2020.4.3)
Requirement already satisfied: appdirs>=1.4.0 in /usr/local/lib/python3.6/dist-
↪packages (from pyopencl->pyqmri) (1.4.4)
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.6/dist-
↪packages (from jupyter-client->ipyparallel->pyqmri) (4.6.3)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/dist-
↪packages (from ipython>=4->ipyparallel->pyqmri) (50.3.2)
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packages␣
↪(from ipython>=4->ipyparallel->pyqmri) (2.6.1)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-packages␣
↪(from ipython>=4->ipyparallel->pyqmri) (0.7.5)
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/
↪python3.6/dist-packages (from ipython>=4->ipyparallel->pyqmri) (4.8.0)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.
↪6/dist-packages (from ipython>=4->ipyparallel->pyqmri) (1.0.18)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/dist-
↪packages (from ipython>=4->ipyparallel->pyqmri) (0.8.1)
Requirement already satisfied: dataclasses>=0.7; python_version <= "3.6" in /usr/
↪local/lib/python3.6/dist-packages (from pytools>=2017.6->pyopencl->pyqmri) (0.7)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dist-
↪packages (from pexpect; sys_platform != "win32"->ipython>=4->ipyparallel->pyqmri)␣
↪(0.6.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages␣
↪(from prompt-toolkit<2.0.0,>=1.0.4->ipython>=4->ipyparallel->pyqmri) (0.2.5)
```

And that was it already. **PyQMRI** is installed on the system. # Example data Next, lets have a look at some example data to run the reconstruction with. We will use *radially acquired VFA* data consisting of 10 *flip angles* (Scans) and 34 *radial projections* each.

Downloading this file might take some time, depending on the speed of the internet connection.

[ ]:
```
import os
%cd sample_data/
if not os.path.isfile("23052018_VFA_34.h5"):
  !wget https://zenodo.org/record/1410918/files/23052018_VFA_34.h5
%cd ..
```

```
/content/sample_data
--2020-11-03 14:46:43--  https://zenodo.org/record/1410918/files/23052018_VFA_34.h5
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1688253784 (1.6G) [application/octet-stream]
Saving to: '23052018_VFA_34.h5'

23052018_VFA_34.h5  100%[===================>]   1.57G  9.43MB/s    in 3m 0s

2020-11-03 14:49:44 (8.93 MB/s) - '23052018_VFA_34.h5' saved [1688253784/1688253784]

/content
```

Ok so we have some data now. Lets explore what is inside of this .h5 file:

[ ]:
```
import h5py

with h5py.File('sample_data/23052018_VFA_34.h5', 'r') as file:
  print("Data content: " + str(file.keys()))
  print("Additional (sequence related) attributes: " + str(file.attrs.keys()))
```

```
Data content: <KeysViewHDF5 ['Coils', 'dcf', 'fa_corr', 'imag_dat', 'imag_traj',
↪'real_dat', 'real_traj']>
Additional (sequence related) attributes: <KeysViewHDF5 ['TR', 'data_normalized_with_
↪dcf', 'flip_angle(s)', 'image_dimensions']>
```

Now we know what to expect. This File reflects the rawdata and necessary attributes to start the reconstruction/fitting process of PyQMRI.

The data was acquired in a non-Cartesian fashion, as can be seen from the fact that *_traj* entries exists in the data itself.

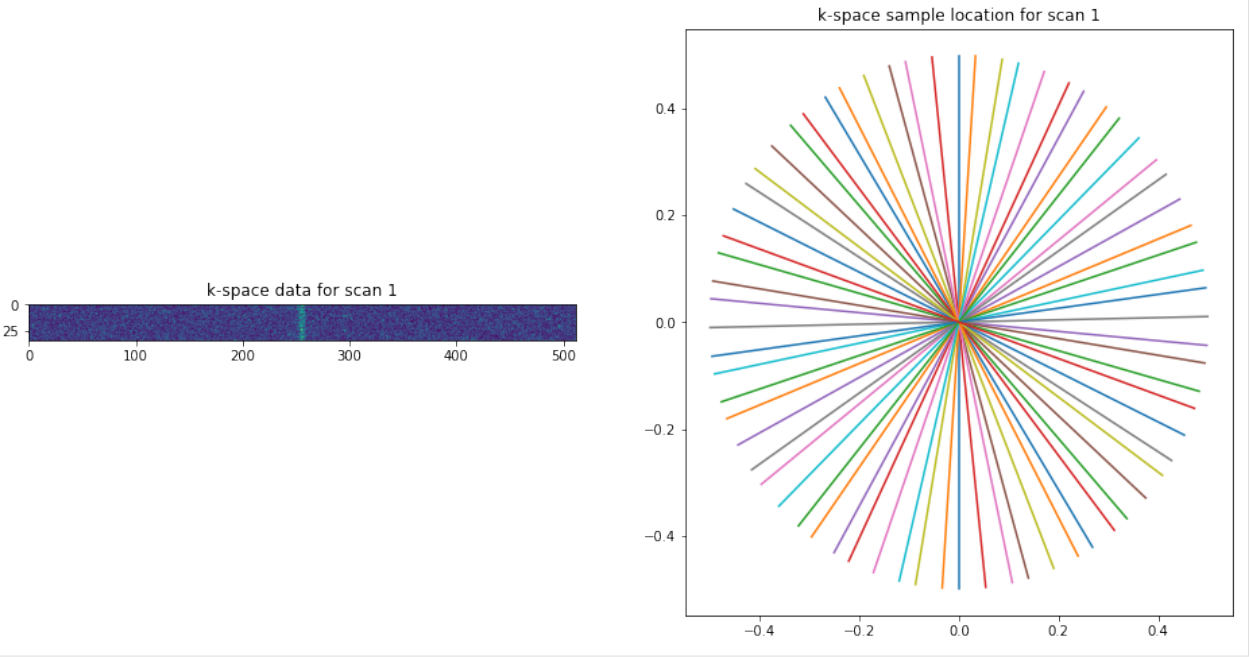
Visualizing the data and trajectory gives:

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

with h5py.File('sample_data/23052018_VFA_34.h5', 'r') as file:
  data = file["real_dat"][()] + 1j* file["imag_dat"][()]
  trajectory = file["real_traj"][()] + 1j* file["imag_traj"][()]

fig = plt.figure(figsize=(16,8))
ax = fig.add_subplot(121)
ax.imshow(np.abs(data[0,0,0]))
tmp = plt.title("k-space data for scan 1")
ax = fig.add_subplot(122)
trajectory.shape
```

(continues on next page)

(continued from previous page)

```
plot = ax.plot(trajectory[0].real.T, trajectory[0].imag.T)
tmp = plt.title("k-space sample location for scan 1")
```



As this file served as input for an older version we need to deleted to old coil sensitivity profiles *Coils* because they will have a wrong orientation.

[ ]:
```
with h5py.File('sample_data/23052018_VFA_34.h5', 'r+') as file:
  del file["Coils"]
```

Lets check if that was succesfull:

[ ]:
```
with h5py.File('sample_data/23052018_VFA_34.h5', 'r') as file:
  print("Data content: " + str(file.keys()))
  print("Additional (sequence related) attributes: " + str(file.attrs.keys()))
```

```
Data content: <KeysViewHDF5 ['dcf', 'fa_corr', 'imag_dat', 'imag_traj', 'real_dat',
↪'real_traj']>
Additional (sequence related) attributes: <KeysViewHDF5 ['TR', 'data_normalized_with_
↪dcf', 'flip_angle(s)', 'image_dimensions']>
```

Perfect. The old coil sensitivity maps are gone and we are ready to start the fitting. As a quick side note, the *dcf* entry is also depracted and will not be used in the current version. We do not need to delete it, as it is not read in at all.

1.2 Starting the fitting with PyQMRI

Now we have two possibilities to start the reconstruction, either we use the command line interface or we start it from within the Python interpreter. But first we need to start an ipython cluster which is used for a distributed estimation of coil sensitivity profiles (each slice separate in a process):

[ ]:
```
!ipcluster start --daemonize
import time
time.sleep(5) # Wait 5 seconds for the cluster to start
```

Lets first have a look at the command line interface. Using *–help* we can see what flags we can pass.

[ ]:
```
!pyqmri --help
```

```
usage: pyqmri [-h] [--recon_type TYPE] [--reg_type REG] [--slices SLICES]
              [--trafo TRAFO] [--streamed STREAMED] [--par_slices PAR_SLICES]
              [--data FILE] [--config CONFIG] [--imagespace IMAGESPACE]
              [--sms SMS] [--OCL_GPU USE_GPU]
              [--devices [DEVICES [DEVICES ...]]] [--dz DZ]
              [--weights [WEIGHTS [WEIGHTS ...]]] [--useCGguess USECG]
              [--out OUTDIR] [--model SIG_MODEL] [--modelfile MODELFILE]
              [--modelname MODELNAME] [--outdir OUTDIR]
              [--double_precision DOUBLE_PRECISION]

T1 quantification from VFA data. By default runs 3D regularization for TGV.

optional arguments:
  -h, --help            show this help message and exit
  --recon_type TYPE     Choose reconstruction type (currently only 3D)
  --reg_type REG        Choose regularization type (default: TGV) options are:
                        TGV, TV, all
  --slices SLICES       Number of reconstructed slices (default=40).
                        Symmetrical around the center slice.
  --trafo TRAFO         Choos between radial (1, default) and Cartesian (0)
                        sampling.
  --streamed STREAMED   Enable streaming of large data arrays (e.g. >10
                        slices).
  --par_slices PAR_SLICES
                        number of slices per package. Volume devided by GPU's
                        and par_slices must be an even number!
  --data FILE           Full path to input data. If not provided, a file
                        dialog will open.
  --config CONFIG       Name of config file to use (assumed to be in the same
                        folder). If not specified, use default parameters.
  --imagespace IMAGESPACE
                        Select if Reco is performed on images (1) or on kspace
                        (0) data. Defaults to 0
  --sms SMS             Switch to SMS reconstruction
  --OCL_GPU USE_GPU     Select if CPU or GPU should be used as OpenCL
                        platform. Defaults to GPU (1). CAVE: CPU FFT not
                        working
  --devices [DEVICES [DEVICES ...]]
                        Device ID of device(s) to use for streaming. -1
                        selects all available devices
  --dz DZ               Ratio of physical Z to X/Y dimension. X/Y is assumed
                        to be isotropic. Defaults to 1
  --weights [WEIGHTS [WEIGHTS ...]]
                        Ratio of unkowns to each other. Defaults to 1. If
                        passed, needs to be in the same size as the number of
                        unknowns
  --useCGguess USECG    Switch between CG sense and simple FFT as initial
                        guess for the images.
  --out OUTDIR          Set output directory. Defaults to the input file
                        directory
```

(continues on next page)

(continued from previous page)

```
  --model SIG_MODEL     Name of the signal model to use. Defaults to VFA.
                        Please put your signal model file in the Model
                        subfolder.
  --modelfile MODELFILE
                        Path to the model file.
  --modelname MODELNAME
                        Name of the model to use.
  --outdir OUTDIR       The path to the output directory. Defaults to the
                        location of the input.
  --double_precision DOUBLE_PRECISION
                        Switch between single (False, default) and double
                        precision (True). Usually, single precision gives high
                        enough accuracy.
```

With that, lets try to start the reconstruction. First we generate a default configuration file in the current folder which we can later edit. We can have a look at the File using pythons configparser:

[ ]:
```
import pyqmri
pyqmri._helper_fun._utils.gen_default_config()

import configparser
config = configparser.ConfigParser()
config.read('default.ini')
print(config.sections())
```

```
['TGV', 'TV']
```

We can see that the file contains two sections *TGV* and *TV* which are used for the regularization associated with each of them. Both are build up the same way and only differ in the values for the regularization strength. Lets examine the TGV part:

[ ]:
```
config.items('TGV')
```

```
[('max_iters', '1000'),
 ('start_iters', '10'),
 ('max_gn_it', '7'),
 ('lambd', '1e2'),
 ('gamma', '5e-2'),
 ('delta', '1e-1'),
 ('omega', '0'),
 ('display_iterations', '0'),
 ('gamma_min', '5e-3'),
 ('delta_max', '1e2'),
 ('omega_min', '0'),
 ('tol', '1e-6'),
 ('stag', '1e10'),
 ('delta_inc', '10'),
 ('gamma_dec', '0.5'),
 ('omega_dec', '0.5')]
```

We can see all kind of optimization related values, e.g. number of iteration to use or regularization strength. For now, lets just enable plotting and save the file:

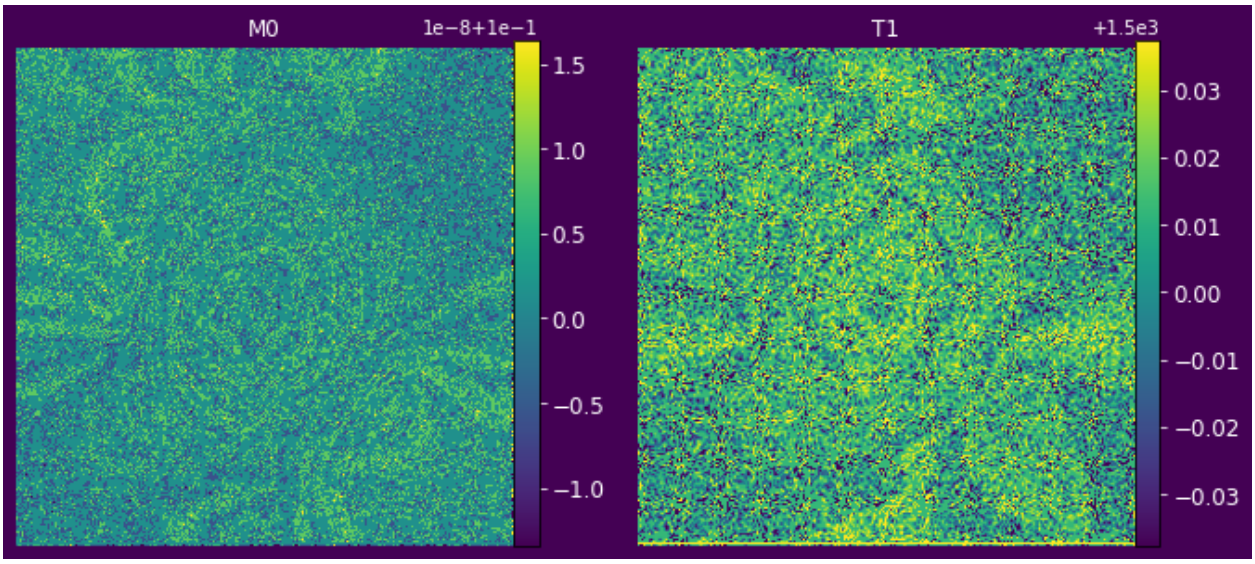
[ ]:
```
config.set('TGV', 'display_iterations', '1')
with open('default.ini', 'w') as f:
  config.write(f)
```

With that we can start a reconstruction/fitting process within Python by:

```
[ ]: pyqmri.run(data='sample_data/23052018_VFA_34.h5', model='VFA', slices=1)
```

```
Unknown command line arguments passed: ['-f', '/root/.local/share/jupyter/runtime/
↪kernel-cfc11e8e-c256-4149-a90f-aabb096a9e7a.json']. These will be ignored for␣
↪fitting.
Using provied flip angle correction.
GPU OpenCL platform <NVIDIA CUDA> found with 1 device(s) and OpenCL-version <OpenCL 1.
↪2 CUDA 10.1.152>
Using single precision
Using single precision
Using single precision
Estimated SNR from kspace 26336.412
Data scale:  314.33432
Using single precision
Initial Norm:  491.0632
Initial Ratio:  [ 25.611603 490.39487 ]
Norm after rescale:  491.06323
Ratio after rescale:  [347.23416 347.23413]
----------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 298.904084
Costs of T(G)V: 0.000000
Costs of L2 Term: 701.095916
----------------------------------------------------------------------------
Function value at GN-Step 0: 1000.000000
----------------------------------------------------------------------------
```

```
/usr/local/lib/python3.6/dist-packages/pyqmri/solver.py:895: RuntimeWarning: divide␣
↪by zero encountered in double_scalars
  self._DTYPE_real(1 / (1 + par[0] / par[3])),
```




```
----------------------------------------------------------------------------
GN-Iter: 0  Elapsed time: 6.671120 seconds
----------------------------------------------------------------------------
Initial Norm:  307.5716
Initial Ratio:  [259.0445  165.81989]
Norm after rescale:  307.5716
Ratio after rescale:  [217.486   217.48593]
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 15.180288
Costs of T(G)V: 0.471494
Costs of L2 Term: 53.523239
----------------------------------------------------------------------
Function value at GN-Step 1: 69.175021
----------------------------------------------------------------------
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0000 ---- Primal: 1.55e+01, Dual: -3.98e+01, Gap: 5.53e+01, Beta: 1.29e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
----------------------------------------------------------------------
GN-Iter: 1  Elapsed time: 7.632198 seconds
----------------------------------------------------------------------
Initial Norm:  244.77806
Initial Ratio:  [229.84364  84.19142]
Norm after rescale:  244.77806
Ratio after rescale:  [173.08423 173.08423]
----------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 3.305516
Costs of T(G)V: 0.255211
Costs of L2 Term: 2.603047
----------------------------------------------------------------------
Function value at GN-Step 2: 6.163775
----------------------------------------------------------------------
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0000 ---- Primal: 3.50e+00, Dual: -3.11e+02, Gap: 3.15e+02, Beta: 1.03e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0010 ---- Primal: 2.71e+00, Dual: -8.88e+01, Gap: 9.15e+01, Beta: 1.04e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0020 ---- Primal: 2.63e+00, Dual: -4.47e+01, Gap: 4.73e+01, Beta: 1.05e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
----------------------------------------------------------------------
GN-Iter: 2  Elapsed time: 14.005931 seconds
----------------------------------------------------------------------
Initial Norm:  264.14438
Initial Ratio:  [210.42711 159.6643 ]
Norm after rescale:  264.1444
Ratio after rescale:  [186.77832 186.77827]
----------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.350412
Costs of T(G)V: 0.136699
Costs of L2 Term: 0.222330
----------------------------------------------------------------------
Function value at GN-Step 3: 2.709440
----------------------------------------------------------------------
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0000 ---- Primal: 2.48e+00, Dual: -8.74e+02, Gap: 8.77e+02, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0010 ---- Primal: 2.59e+00, Dual: -2.72e+03, Gap: 2.72e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0020 ---- Primal: 2.51e+00, Dual: -7.70e+02, Gap: 7.73e+02, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Terminated at iteration 30 because the energy decrease in the primal problem was less␣
↪than 4.822e-07
------------------------------------------------------------------------
GN-Iter: 3  Elapsed time: 11.410784 seconds
------------------------------------------------------------------------
Initial Norm:  275.24033
Initial Ratio:  [194.6561 194.5925]
Norm after rescale:  275.24033
Ratio after rescale:  [194.6243  194.62431]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.255412
Costs of T(G)V: 0.167106
Costs of L2 Term: 0.222241
------------------------------------------------------------------------
Function value at GN-Step 4: 2.644759
------------------------------------------------------------------------
<Figure size 432x288 with 0 Axes>
Iteration: 0000 ---- Primal: 2.40e+00, Dual: -1.91e+03, Gap: 1.92e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0010 ---- Primal: 2.55e+00, Dual: -8.48e+02, Gap: 8.51e+02, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0020 ---- Primal: 2.44e+00, Dual: -5.57e+02, Gap: 5.59e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0030 ---- Primal: 2.41e+00, Dual: -6.08e+02, Gap: 6.10e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0040 ---- Primal: 2.40e+00, Dual: -6.97e+02, Gap: 6.99e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0050 ---- Primal: 2.39e+00, Dual: -6.20e+02, Gap: 6.23e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0060 ---- Primal: 2.39e+00, Dual: -6.27e+02, Gap: 6.29e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0070 ---- Primal: 2.39e+00, Dual: -6.33e+02, Gap: 6.35e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0080 ---- Primal: 2.38e+00, Dual: -6.38e+02, Gap: 6.40e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0090 ---- Primal: 2.38e+00, Dual: -6.40e+02, Gap: 6.42e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0100 ---- Primal: 2.38e+00, Dual: -6.40e+02, Gap: 6.42e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0110 ---- Primal: 2.38e+00, Dual: -6.41e+02, Gap: 6.43e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0120 ---- Primal: 2.37e+00, Dual: -6.41e+02, Gap: 6.43e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0130 ---- Primal: 2.37e+00, Dual: -6.41e+02, Gap: 6.44e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0140 ---- Primal: 2.37e+00, Dual: -6.42e+02, Gap: 6.44e+02, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
------------------------------------------------------------------------
GN-Iter: 4  Elapsed time: 53.036350 seconds
------------------------------------------------------------------------
Initial Norm:  311.4288
Initial Ratio:  [241.32141 196.85497]
Norm after rescale:  311.4288
Ratio after rescale:  [220.21347 220.21338]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.256774
Costs of T(G)V: 0.136053
Costs of L2 Term: 0.198378
------------------------------------------------------------------------
Function value at GN-Step 5: 2.591205
------------------------------------------------------------------------
<Figure size 432x288 with 0 Axes>
Iteration: 0000 ---- Primal: 2.39e+00, Dual: -3.78e+03, Gap: 3.78e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0010 ---- Primal: 2.70e+00, Dual: -4.10e+03, Gap: 4.10e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0020 ---- Primal: 2.48e+00, Dual: -3.57e+03, Gap: 3.57e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0030 ---- Primal: 2.45e+00, Dual: -1.25e+03, Gap: 1.25e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0040 ---- Primal: 2.41e+00, Dual: -1.12e+03, Gap: 1.12e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0050 ---- Primal: 2.40e+00, Dual: -1.14e+03, Gap: 1.15e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0060 ---- Primal: 2.40e+00, Dual: -1.18e+03, Gap: 1.18e+03, Beta: 1.01e+03
```

```
<Figure size 432x288 with 0 Axes>
Terminated at iteration 70 because the energy decrease in the primal problem was less␣
↪than 1.929e-07
------------------------------------------------------------------------
GN-Iter: 5  Elapsed time: 24.623435 seconds
------------------------------------------------------------------------
Initial Norm:  314.40265
Initial Ratio:  [225.6203  218.96234]
Norm after rescale:  314.40262
Ratio after rescale:  [222.31625 222.31622]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.230266
Costs of T(G)V: 0.163726
Costs of L2 Term: 0.195346
------------------------------------------------------------------------
Function value at GN-Step 6: 2.589338
------------------------------------------------------------------------
<Figure size 432x288 with 0 Axes>
Iteration: 0000 ---- Primal: 2.38e+00, Dual: -1.17e+03, Gap: 1.17e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0010 ---- Primal: 2.52e+00, Dual: -2.88e+03, Gap: 2.88e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0020 ---- Primal: 2.43e+00, Dual: -1.99e+03, Gap: 2.00e+03, Beta: 1.00e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0030 ---- Primal: 2.42e+00, Dual: -1.53e+03, Gap: 1.53e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0040 ---- Primal: 2.40e+00, Dual: -1.34e+03, Gap: 1.34e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0050 ---- Primal: 2.40e+00, Dual: -1.14e+03, Gap: 1.14e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0060 ---- Primal: 2.39e+00, Dual: -1.15e+03, Gap: 1.15e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0070 ---- Primal: 2.39e+00, Dual: -1.18e+03, Gap: 1.18e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0080 ---- Primal: 2.39e+00, Dual: -1.18e+03, Gap: 1.18e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0090 ---- Primal: 2.38e+00, Dual: -1.20e+03, Gap: 1.20e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0100 ---- Primal: 2.38e+00, Dual: -1.20e+03, Gap: 1.20e+03, Beta: 1.01e+03
<Figure size 432x288 with 0 Axes>
Iteration: 0110 ---- Primal: 2.38e+00, Dual: -1.20e+03, Gap: 1.20e+03, Beta: 1.01e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
Iteration: 0120 ---- Primal: 2.38e+00, Dual: -1.20e+03, Gap: 1.20e+03, Beta: 1.01e+03
```

```
<Figure size 432x288 with 0 Axes>
```

```
Terminated at iteration 130 because the energy decrease in the primal problem was␣
↪less than 9.148e-07
----------------------------------------------------------------------------
GN-Iter: 6  Elapsed time: 43.807075 seconds
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.230266
Costs of T(G)V: 0.150245
Costs of L2 Term: 0.195236
----------------------------------------------------------------------------
Function value at GN-Step 7: 2.575747
----------------------------------------------------------------------------
```

or directly from the command line via:

```
[ ]: !pyqmri --data sample_data/23052018_VFA_34.h5 --trafo 1 --slices 1 --reg_type TGV --
↪config default --model VFA
```

```
Using provied flip angle correction.
GPU OpenCL platform <NVIDIA CUDA> found with 1 device(s) and OpenCL-version <OpenCL 1.
↪2 CUDA 10.1.152>
Using precomputed coil sensitivities
Using precomputed images
Estimated SNR from kspace 26336.412
Data scale:  314.33432
Using single precision
Initial Norm:  491.0632
Initial Ratio:  [ 25.611603 490.39487 ]
Norm after rescale:  491.06323
Ratio after rescale:  [347.23416 347.23413]
----------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 298.904084
Costs of T(G)V: 0.000000
Costs of L2 Term: 701.095916
----------------------------------------------------------------------------
Function value at GN-Step 0: 1000.000000
----------------------------------------------------------------------------
/usr/local/lib/python3.6/dist-packages/pyqmri/solver.py:895: RuntimeWarning: divide␣
↪by zero encountered in double_scalars
  self._DTYPE_real(1 / (1 + par[0] / par[3])),
<Figure size 1200x600 with 8 Axes>
/usr/local/lib/python3.6/dist-packages/pyqmri/solver.py:895: RuntimeWarning: divide␣
↪by zero encountered in double_scalars
  self._DTYPE_real(1 / (1 + par[0] / par[3])),
----------------------------------------------------------------------------
GN-Iter: 0  Elapsed time: 4.828518 seconds
----------------------------------------------------------------------------
Initial Norm:  307.57156
Initial Ratio:  [259.0446  165.81972]
Norm after rescale:  307.57153
Ratio after rescale:  [217.48592 217.4859 ]
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 15.181189
Costs of T(G)V: 0.471586
Costs of L2 Term: 53.523186
------------------------------------------------------------------------
Function value at GN-Step 1: 69.175961
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
------------------------------------------------------------------------
GN-Iter: 1  Elapsed time: 7.731095 seconds
------------------------------------------------------------------------
Initial Norm:  244.77844
Initial Ratio:  [229.84421  84.19102]
Norm after rescale:  244.77843
Ratio after rescale:  [173.08449 173.08449]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 3.305495
Costs of T(G)V: 0.255196
Costs of L2 Term: 2.603028
------------------------------------------------------------------------
Function value at GN-Step 2: 6.163719
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
------------------------------------------------------------------------
GN-Iter: 2  Elapsed time: 14.126581 seconds
------------------------------------------------------------------------
Initial Norm:  264.2159
Initial Ratio:  [210.53906 159.63506]
Norm after rescale:  264.2159
Ratio after rescale:  [186.82884 186.82887]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.351163
Costs of T(G)V: 0.136595
Costs of L2 Term: 0.222226
------------------------------------------------------------------------
Function value at GN-Step 3: 2.709984
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
------------------------------------------------------------------------
GN-Iter: 3  Elapsed time: 27.063896 seconds
------------------------------------------------------------------------
Initial Norm:  284.62256
Initial Ratio:  [205.68614 196.73132]
```

(continues on next page)

(continued from previous page)

```
Norm after rescale:  284.62253
Ratio after rescale:  [201.25853 201.2585 ]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.255787
Costs of T(G)V: 0.146302
Costs of L2 Term: 0.217105
------------------------------------------------------------------------
Function value at GN-Step 4: 2.619195
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 140 because the energy decrease in the primal problem was␣
↪less than 8.718e-07
------------------------------------------------------------------------
GN-Iter: 4  Elapsed time: 46.654978 seconds
------------------------------------------------------------------------
Initial Norm:  305.51443
Initial Ratio:  [230.03995 201.04901]
Norm after rescale:  305.51443
Ratio after rescale:  [216.03131 216.03133]
------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.235969
Costs of T(G)V: 0.148358
Costs of L2 Term: 0.201862
------------------------------------------------------------------------
Function value at GN-Step 5: 2.586189
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 130 because the energy decrease in the primal problem was␣
↪less than 9.498e-07
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------------
GN-Iter: 5  Elapsed time: 43.591899 seconds
------------------------------------------------------------------------------
Initial Norm:  316.39548
Initial Ratio:  [231.47491 215.6976 ]
Norm after rescale:  316.3955
Ratio after rescale:  [223.7254  223.72539]
------------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.230008
Costs of T(G)V: 0.154566
Costs of L2 Term: 0.194509
------------------------------------------------------------------------------
Function value at GN-Step 6: 2.579083
------------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 130 because the energy decrease in the primal problem was␣
↪less than 9.502e-07
------------------------------------------------------------------------------
GN-Iter: 6  Elapsed time: 43.606140 seconds
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Initial Cost: 2569.238114
Costs of Data: 2.230008
Costs of T(G)V: 0.151465
Costs of L2 Term: 0.194400
------------------------------------------------------------------------------
Function value at GN-Step 7: 2.575873
------------------------------------------------------------------------------
```

1.3 PyQMRI text-based models

In this section we will have a look at the text-based models. First lets generate a default model-file and examine its content:

[ ]:
```
pyqmri.generate_text_models()

with open('models.ini','r') as file:
  for line in file.readlines():
    print(line)
```

```
[MonoExp]

parameter = TE

unknowns = M0 A1

signal = M0*exp(-TE*A1)

box_constraints_lower = 0,0

box_constraints_upper = 100,1

real_value_constraints = False,True

guess = 1,0.01

rescale = M0,1/A1

estimate_individual_phase = False


[VFA-E1]

parameter = TR fa fa_corr

unknowns = M0 E_1

signal = M0*sin(fa*fa_corr)*(1-E_1)/(1-E_1*cos(fa*fa_corr))

box_constraints_lower = 0,0.9048

box_constraints_upper = 10,0.99909

real_value_constraints = False,True

guess = 1,0.99667

rescale = M0,-TR/log(E_1)

estimate_individual_phase = False
```

We see that, similar to the config file, each model is defined via a section, e.g. *[VFA-E1]* for the VFA model. Next the *parameters* are describe sequence related parameters that are known a priori. The *unknowns* determine the parameters which should be fit and *signal* describes the relation between signal intensity and sequence/parameters. In additon, boxconstrains and real value constrains can be set for each unknown separately. The initial guess *guess* is meant as constant value for the whole image. *resacle* devines rescaling functions, if only the unknonw itself is given, no additional transformation is used. Finally, *estimate_individual_phase* can be used to estimate phase variations between each scan, e.g. usefully in diffusion imaging.

Now we also copy the "flip_angle(s)" attribute to "fa". This is necessary because all parameters defined in the model file need to be named the same way in the data attributes section

[ ]:
```
import numpy as np
```

(continues on next page)

(continued from previous page)

```
with h5py.File('sample_data/23052018_VFA_34.h5', 'r+') as file:
  file.attrs["fa"] = file.attrs["flip_angle(s)"]/180*np.pi
```

Lets take a look now at how to invoke the model file, for simplicity, we just use the CLI interface here. Steps from within Python are identical.

```
[ ]: !pyqmri --data sample_data/23052018_VFA_34.h5 --trafo 1 --slices 1 --reg_type TGV --
     ↪config default --modelfile models.ini --modelname VFA-E1
```

```
Using provied flip angle correction.
GPU OpenCL platform <NVIDIA CUDA> found with 1 device(s) and OpenCL-version <OpenCL 1.
↪2 CUDA 10.1.152>
Using precomputed coil sensitivities
Using precomputed images
Estimated SNR from kspace 26336.412
Data scale:  314.33432
Using single precision
Initial Norm:  5064.0312
Initial Ratio:  [  24.441225 5063.972   ]
Norm after rescale:  5064.0317
Ratio after rescale:  [3580.811 3580.811]
----------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 978.795790
Costs of T(G)V: 0.000000
Costs of L2 Term: 21.204210
----------------------------------------------------------------------------
Function value at GN-Step 0: 1000.000000
----------------------------------------------------------------------------
/usr/local/lib/python3.6/dist-packages/pyqmri/solver.py:895: RuntimeWarning: divide␣
↪by zero encountered in double_scalars
  self._DTYPE_real(1 / (1 + par[0] / par[3])),
<Figure size 1200x600 with 8 Axes>
/usr/local/lib/python3.6/dist-packages/pyqmri/solver.py:895: RuntimeWarning: divide␣
↪by zero encountered in double_scalars
  self._DTYPE_real(1 / (1 + par[0] / par[3])),
----------------------------------------------------------------------------
GN-Iter: 0  Elapsed time: 5.519405 seconds
----------------------------------------------------------------------------
Initial Norm:  2885.6672
Initial Ratio:  [2244.1272 1814.1027]
Norm after rescale:  2885.6672
Ratio after rescale:  [2040.4751 2040.4749]
----------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 51.949737
Costs of T(G)V: 0.004095
Costs of L2 Term: 1.888338
----------------------------------------------------------------------------
Function value at GN-Step 1: 53.842169
----------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
----------------------------------------------------------------------------
GN-Iter: 1  Elapsed time: 8.384667 seconds
----------------------------------------------------------------------------
Initial Norm:  2167.455
```

(continues on next page)

(continued from previous page)

```
Initial Ratio:  [2166.235     72.71026]
Norm after rescale:  2167.455
Ratio after rescale:  [1532.6222 1532.6223]
----------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 0.721448
Costs of T(G)V: 0.003955
Costs of L2 Term: 0.008957
----------------------------------------------------------------------------
Function value at GN-Step 2: 0.734360
----------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
----------------------------------------------------------------------------
GN-Iter: 2  Elapsed time: 14.788510 seconds
----------------------------------------------------------------------------
Initial Norm:  3601.9456
Initial Ratio:  [1655.7506 3198.8281]
Norm after rescale:  3601.9453
Ratio after rescale:  [2546.9604 2546.9595]
----------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 0.262864
Costs of T(G)V: 0.002272
Costs of L2 Term: 0.001125
----------------------------------------------------------------------------
Function value at GN-Step 3: 0.266261
----------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
----------------------------------------------------------------------------
GN-Iter: 3  Elapsed time: 27.715470 seconds
----------------------------------------------------------------------------
Initial Norm:  3436.7874
Initial Ratio:  [3010.1782 1658.4136]
Norm after rescale:  3436.787
Ratio after rescale:  [2430.1753 2430.1758]
----------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 1.659800
Costs of T(G)V: 0.002708
Costs of L2 Term: 0.000767
----------------------------------------------------------------------------
Function value at GN-Step 4: 1.663276
----------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```

(continues on next page)

(continued from previous page)

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 130 because the energy decrease in the primal problem was␣
↪less than 3.046e-07
------------------------------------------------------------------------
GN-Iter: 4  Elapsed time: 44.388520 seconds
------------------------------------------------------------------------
Initial Norm:  3459.8164
Initial Ratio:  [2625.2764 2253.4983]
Norm after rescale:  3459.8167
Ratio after rescale:  [2446.4595 2446.4602]
------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 0.168425
Costs of T(G)V: 0.002520
Costs of L2 Term: 0.000706
------------------------------------------------------------------------
Function value at GN-Step 5: 0.171652
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 50 because the energy decrease in the primal problem was less␣
↪than 8.827e-07
------------------------------------------------------------------------
GN-Iter: 5  Elapsed time: 18.530759 seconds
------------------------------------------------------------------------
Initial Norm:  3418.1575
Initial Ratio:  [2693.4111 2104.5989]
Norm after rescale:  3418.158
Ratio after rescale:  [2417.0027 2417.0024]
------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 0.157330
Costs of T(G)V: 0.002076
Costs of L2 Term: 0.000615
------------------------------------------------------------------------
Function value at GN-Step 6: 0.160021
------------------------------------------------------------------------
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
Terminated at iteration 40 because the energy decrease in the primal problem was less␣
↪than 4.501e-07
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------------
GN-Iter: 6  Elapsed time: 15.461497 seconds
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Initial Cost: 85085.258519
Costs of Data: 0.157330
Costs of T(G)V: 0.001422
Costs of L2 Term: 0.000615
------------------------------------------------------------------------------
Function value at GN-Step 7: 0.159367
------------------------------------------------------------------------------
```

1.4 Displaying the results

Finally, lets have a quick look at the results. These are stored in the PyQMRI_out folder next to the input data per default and named after the used model. All reconstructions feature a time stamp to easily distinguis them.

[ ]:
```
import os

outdirs_VFA = os.listdir('sample_data/PyQMRI_out/VFA')
outdirs_VFA.sort()

# Lets have a closer look at the most recent one:

files = os.listdir('sample_data/PyQMRI_out/VFA/'+outdirs_VFA[-1])

print(files)
print(outdirs_VFA)
```

```
['config.ini', 'output_23052018_VFA_34.h5']
['2020-11-03  14-43-47', '2020-11-03  14-44-19', '2020-11-03  14-49-53', '2020-11-03␣
↪14-52-56']
```

We see that there is a config.ini file along with the output. The config file contains all the parameters used in the reconstruction for this specific dataset. Now we take a closer look at the output:

[ ]:
```
with h5py.File('sample_data/PyQMRI_out/VFA/'+outdirs_VFA[-1]+os.sep+files[-1],'r') as␣
↪file:
  print(file.keys())
```

```
<KeysViewHDF5 ['images_ifft', 'tgv_result_iter_0', 'tgv_result_iter_1', 'tgv_result_
↪iter_2', 'tgv_result_iter_3', 'tgv_result_iter_4', 'tgv_result_iter_5', 'tgv_result_
↪iter_6']>
```

We can see that the file containts several entries. *images_ifft* amounts to iamges based on a simple image reconstruction of the rawdata (either CG-SENSE based or just iFFT). In addition, we have a *tgv_result_iter_x* for each Gauss-Newton iteration of the optimization algorithm.

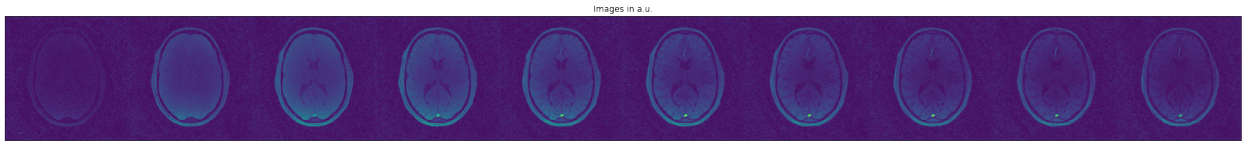
As final step, we can visualize the fitting results for this example (*tgv_result_iter_6*) and the initial images after CG-SENSE reconstruction:

[ ]:
```
with h5py.File('sample_data/PyQMRI_out/VFA/'+outdirs_VFA[-1]+os.sep+files[-1],'r') as␣
↪file:
  result = file["tgv_result_iter_6"][()]
  images = file["images_ifft"][()]
```

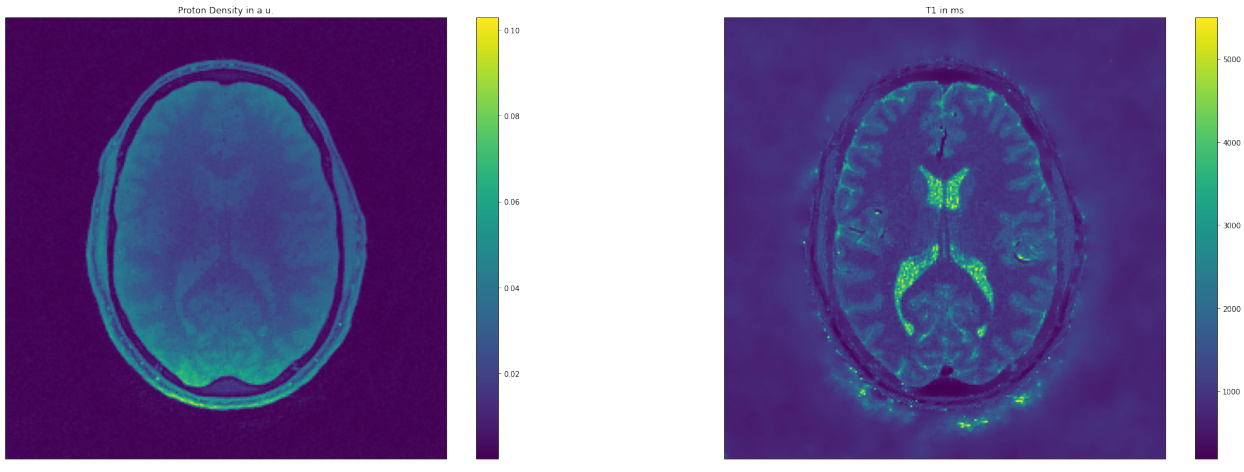
(continues on next page)

(continued from previous page)

```
%matplotlib inline

import matplotlib.pyplot as plt

num_images = images.shape[0]
images = np.swapaxes(images, -2, -1)
images = images.reshape(images.shape[-2]*num_images, -1).T
fig = plt.figure(figsize=(32,4))
ax = fig.add_subplot(111)
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.imshow(np.abs(np.squeeze(images)))
plt.title('Images in a.u.')
plt.show()
```



[ ]:
```
fig = plt.figure(figsize=(32,11))
ax = fig.add_subplot(121)
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
M0_plot = plt.imshow(np.abs(np.squeeze(result[0])))
cb = plt.colorbar(M0_plot)
tmp = plt.title('Proton Density in a.u.')
ax = fig.add_subplot(122)
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
T1_plot = plt.imshow(np.abs(np.squeeze(result[1])))
cb = plt.colorbar(T1_plot)
tmp = plt.title('T1 in ms')
```



CHAPTER 2

Installation Guide

First make sure that you have a working OpenCL installation

- OpenCL is usually shipped with GPU driver (Nvidia/AMD)
- Install the ocl_icd and the OpenCL-Headers

  ```
  apt-get install ocl_icd* opencl-headers
  ```

Possible restart of system after installing new drivers and check if OpenCL is working

- Build clinfo:
- Run clinfo in terminal and check for errors

Install clFFT library:

- Either use the package repository,e.g.:

  ```
  apt-get install libclfft*
  ```

- Or download a prebuild binary of clfft
  - Please refer to the clfft docs regarding building
  - If build from source symlink clfft libraries from lib64 to the lib folder and run `ldconfig`

Install gpyfft by following the instruction on the GitHub page.

To Install PyQMRI, a simple

```
pip install pyqmri
```

should be sufficient to install the latest release.

Alternatively, clone the git repository and navigate to the root directory of PyQMRI. Typing

```
pip install .
```

should take care of the other dependencies using PyPI and install the package.

In case OCL > 1.2 is present, e.g. by some CPU driver, and NVidia GPUs needs to be used the flag PRETENED_OCL 1.2 has to be passed to PyOpenCL during the build process. This can be done by:

```
./configure.py --cl-pretend-version=1.2
rm -Rf build
python setup.py install
```

CHAPTER 3

Running the reconstruction

First, start an ipcluster for speeding up the coil sensitivity estimation:

```
ipcluster start -n N
```

where N amounts to the number of processe to be used. If -n N is ommited, as many processes as number of CPU cores available are started.

Reconstruction of the parameter maps can be started either using the terminal by typing:

```
pyqmri
```

or from python by:

```
import pyqmri
pyqmri.run()
```

A list of accepted flags can be printed using

```
pyqmri -h
```

or by fewing the documentation of pyqmri.pyqmri in python.

If reconstructing fewer slices from the volume than acquired, slices will be picked symmetrically from the center of the volume. E.g. reconstructing only a single slice will reconstruct the center slice of the volume.

CHAPTER 4

Prerequests on the .h5 file

The toolbox expects a .h5 file with a certain structure.

- kspace data (assumed to be 5D for VFA) and passed as:
  - real_dat (Scans, Coils, Slices, Projections, Samples)
  - imag_dat (Scans, Coils, Slices, Projections, Samples)

  If radial sampling is used the trajectory is expected to be:
  - real_traj (Scans, Projections, Samples)
  - imag_traj (Scans, Projections, Samples)

  Density compensation is performed internally assuming a simple ramp.
  For Cartesian data Projections and Samples are replaced by ky and kx encodings points and no trajectory is needed.
  Data is assumed to be 2D stack-of-stars, i.e. already Fourier transformed along the fully sampled z-direction.

- flip angle correction (optional) can be passed as:
  - fa_corr (Scans, Coils, Slices, dimY, dimX)
- The image dimension for the full dataset is passed as attribute consiting of:
  - image_dimensions = (dimX, dimY, NSlice)
- Parameters specific to the used model (e.g. TR or flip angle) need to be set as attributes e.g.:
  - TR = 5.38
  - flip_angle(s) = (1,3,5,7,9,11,13,15,17,19)

The specific structure is determined according to the Model file.

If predetermined coil sensitivity maps are available they can be passed as complex dataset, which can saved bedirectly using Python. Matlab users would need to write/use low level hdf5 functions to save a complex array to .h5 file. Coil

sensitivities are assumed to have the same number of slices as the original volume and are intesity normalized. The corresponding .h5 entry is named “Coils”. If no “Coils” parameter is found or the number of “Coil” slices is less than the number of reconstructed slices, the coil sensitivities are determined using the NLINV algorithm and saved into the file.

CHAPTER 5

The config file (*.ini)

A default config file will be generated if no path to a config file is passed as an argument or if no default.ini file is present in the current working directory. After the initial generation the values can be altered to influence regularization or the number of iterations. Seperate values for TV and TGV regularization can be used.

- max_iters: Maximum primal-dual (PD) iterations
- start_iters: PD iterations in the first Gauss-Newton step
- max_gn_it: Maximum number of Gauss Newton iterations
- lambd: Data weighting
- gamma: TGV weighting
- delta: L2-step-penalty weighting (inversely weighted)
- omega: optional H1 regularization (should be set to 0 if no H1 is used)
- display_iterations: Flag for displaying grafical output
- gamma_min: Minimum TGV weighting
- delta_max: Maximum L2-step-penalty weighting
- omega_min: Minimum H1 weighting (should be set to 0 if no H1 is used)
- tol: relative convergence toleranze for PD and Gauss-Newton iterations
- stag: optional stagnation detection between successive PD steps
- delta_inc: Increase factor for delta after each GN step
- gamma_dec: Decrease factor for gamma after each GN step
- omega_dec: Decrease factor for omega after each GN step
- beta: The initial ratio between primal and dual step size of the PD algorithm. Will be adapted during the linesearch.

CHAPTER 6

API Documentation

6.1 PyQMRI

3D model based parameter quantification for MRI.

PyQMRI is a Python module to quantify tissue parameters given a set of MRI measurements, specifically desinged to quantify the parameter of interest. Examples include T1 quantification from variable flip angle or inversion-recovery Look-Locker data, T2 quantification using a mono-exponential fit, or Diffusion Tensor quantification. In addition, a Genereal Model exists that can be invoked using a text file containing the analytical signal equation.

See https://pyqmri.readthedocs.io/en/latest/ for a complete documentation.

Module handling the start up of the fitting procedure.

pyqmri.pyqmri.**run**(*reg_type='TGV'*, *slices=1*, *trafo=True*, *streamed=False*, *par_slices=1*, *data=''*, *model='GeneralModel'*, *config='default'*, *imagespace=False*, *sms=False*, *use_GPU=True*, *devices=0*, *dz=1*, *weights=-1*, *useCGguess=True*, *out=''*, *modelfile='models.ini'*, *modelname='VFA-E1'*, *double_precision=False*, *coils3D=False*, *is3Ddata=False*)

Start a 3D model based reconstruction.

Start a 3D model based reconstruction. If no data path is given, a file dialog can be used to select data at start up. If no other parameters are passed, T1 from a single slice of radially acquired variable flip angle data will be quantified.

If no config file is passed, a default one will be generated in the current folder, the script is run in.

Parameters

- **reg_type** (*str, TGV*) – TGV or TV, defaults to TGV
- **slices** (*int, 1*) – The number of slices to reconsturct. Slices are picked symmetrically from the volume center. Pass -1 to select all slices available. Defaults to 1
- **trafo** (*bool, True*) – Choos between Radial (1) or Cartesian (0) FFT
- **streamed** (*bool, False*) – Toggle between streaming slices to the GPU (1) or computing everything with a single memory transfer (0). Defaults to 0

- **par_slices** (*int, 1*) – Number of slices per streamed package. Volume devided by GPU's and par_slices must be an even number! Defaults to 1
- **data** (*str, ''*) – The path to the .h5 file containing the data to reconstruct. If left empty, a GUI will open and asks for data file selection. This is also the default behaviour.
- **model** (*str, GeneralModel*) – The name of the model which should be used to fit the data. Defaults to 'VFA'. A path to your own model file can be passed. See the Model Class for further information on how to setup your own model.
- **config** (*str, default*) – The path to the confi gfile used for the IRGN reconstruction. If not specified the default config file will be used. If no default config file is present in the current working directory one will be generated.
- **imagespace** (*bool, False*) – Select between fitting in imagespace (1) or in k-space (0). Defaults to 0
- **sms** (*bool, False*) – use Simultaneous Multi Slice Recon (1) or normal reconstruction (0). Defaults to 0
- **devices** (*list of int, 0*) – The device ID of device(s) to use for streaming/reconstruction
- **dz** (*float, 1*) – Ratio of physical Z to X/Y dimension. X/Y is assumed to be isotropic.
- **useCGguess** (*bool, True*) – Switch between CG sense and simple FFT as initial guess for the images.
- **out** (*str, ''*) – Output directory. Defaults to the location of the input file.
- **modelpath** (*str, models.ini*) – Path to the .mod file for the generative model.
- **modelname** (*str, VFA-E1*) – Name of the model in the .mod file to use.
- **weights** (*list of float, -1*) – Optional weights for each unknown. Defaults to -1, i.e. no additional weights are used.
- **double_precision** (*bool, False*) – Enable double precission computation.

6.2 Solver

Module holding the classes for different numerical Optimizer.

class pyqmri.solver.**CGSolver**(*par*, *NScan=1*, *trafo=1*, *SMS=0*)

Conjugate Gradient Optimization Algorithm.

This Class performs a CG reconstruction on single precission complex input data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **NScan** (*int*) – Number of Scan which should be used internally. Do not need to be the same number as in par["NScan"]
- **trafo** (*bool*) – Switch between radial (1) and Cartesian (0) fft.
- **SMS** (*bool*) – Simultaneouos Multi Slice. Switch between noraml (0) and slice accelerated (1) reconstruction.

eval_fwd_kspace_cg(*y*, *x*, *wait_for=None*)
Apply forward operator for image reconstruction. :param y: The result of the computation :type y: PyOpenCL.Array :param x: The input array :type x: PyOpenCL.Array :param wait_for: A List of PyOpenCL events to wait for. :type wait_for: list of PyopenCL.Event, None

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

run(*data*, *iters=30*, *lambd=1e-05*, *tol=1e-08*, *guess=None*, *scan_offset=0*)
Start the CG reconstruction.

All attributes after data are considered keyword only.

Parameters

- **data** (*numpy.array*) – The complex k-space data which serves as the basis for the images.
- **iters** (*int*) – Maximum number of CG iterations
- **lambd** (*float*) – Weighting parameter for the Tikhonov regularization
- **tol** (*float*) – Termination criterion. If the energy decreases below this threshold the algorithm is terminated.
- **guess** (*numpy.array*) – An optional initial guess for the images. If None, zeros is used.

Returns The result of the image reconstruction.

Return type numpy.Array

class pyqmri.solver.**CGSolver_H1**(*prg*, *queue*, *par*, *irgn_par*, *coils*, *linops*)
Conjugate Gradient Optimization Algorithm.

This Class performs a CG reconstruction on single precission complex input data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **prg** (*PyOpenCL Program A PyOpenCL Program containing the*) – kernels for optimization.
- **linops** (*PyQMRI Operator The operator to traverse from*) – parameter to data space.
- **coils** (*PyOpenCL Buffer or empty list*) – coil buffer, empty list if image based fitting is used.

power_iteration(*x*, *num_simulations=50*)

power_iteration_grad(*x*, *num_simulations=50*)

run(*guess*, *data*, *iters=30*)
Start the CG reconstruction.

All attributes after data are considered keyword only.

Parameters

- **guess** (*numpy.array*) – An optional initial guess for the images. If None, zeros is used.
- **data** (*numpy.array*) – The complex k-space data which serves as the basis for the images.
- **iters** (*int*) – Maximum number of CG iterations

Returns The result of the fitting.

Return type dict of numpy.Array

setFvalInit(*fval*)
Set the initial value of the cost function.

Parameters **fval** (*float*) – The initial cost of the optimization problem

updateRegPar(*irgn_par*)
Update the regularization parameters.

Performs an update of the regularization parameters as these usually vary from one to another Gauss-Newton step.

Parameters **(dic)** (*irgn_par*) –

update_box(*outp*, *inp*, *par*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Primal update of the x variable in the Primal-Dual Algorithm.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

class pyqmri.solver.**PDBaseSolver**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *coil*, *model*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
Primal Dual splitting optimization.

This Class performs a primal-dual variable splitting based reconstruction on single precission complex input data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimate of the initial step size based on the operator norm of the linear operator.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL Program A PyOpenCL Program containing the*) – kernels for optimization.
- **reg_type** (*string String to choose between "TV" and "TGV"*) – optimization.
- **data_operator** (*PyQMRI Operator The operator to traverse from*) – parameter to data space.
- **coil** (*PyOpenCL Buffer or empty list*) – coil buffer, empty list if image based fitting is used.
- **model** (*PyQMRI.Model*) – Instance of a PyQMRI.Model to perform plotting

delta
Regularization parameter for L2 penalty on linearization point.

Type float

omega
Not used. Should be set to 0

Type float

lambd
Regularization parameter in front of data fidelity term.

Type float

tol
Relative toleraze to stop iterating

Type float

stag
Stagnation detection parameter

Type float

display_iterations
Switch between plotting (true) of intermediate results

Type bool

mu
Strong convecity parameter (inverse of delta).

Type float

tau
Estimated step size based on operator norm of regularization.

Type float

beta_line
Ratio between dual and primal step size

Type float

theta_line
Line search parameter

Type float

unknwons_TGV
Number of T(G)V unknowns

Type int

unknowns_H1
Number of H1 unknowns (should be 0 for now)

Type int

unknowns
Total number of unknowns (T(G)V+H1)

Type int

num_dev
Total number of compute devices

Type int

dz
Ratio between 3rd dimension and isotropic 1st and 2nd image dimension.

Type float

model
The model which should be fitted

Type PyQMRI.Model

modelgrad
The partial derivatives evaluated at the linearization point. This variable is set in the PyQMRI.irgn Class.

Type PyOpenCL.Array or numpy.Array

min_const
list of minimal values, one for each unknown

Type list of float

max_const
list of maximal values, one for each unknown

Type list of float

real_const
list if a unknown is constrained to real values only. (1 True, 0 False)

Type list of int

static factory(*prg*, *queue*, *par*, *irgn_par*, *init_fval*, *coils*, *linops*, *model*, *reg_type='TGV'*, *SMS=False*, *streamed=False*, *imagespace=False*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Generate a PDSolver object.

Parameters

- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **init_fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **model** (*PyQMRI.Model*) – The model which should be fitted
- **reg_type** (*string, "TGV"*) – String to choose between "TV" and "TGV" optimization.
- **SMS** (*bool, false*) – Switch between standard (false) and SMS (True) fitting.
- **streamed** (*bool, false*) – Switch between streamed (1) and normal (0) reconstruction.
- **imagespace** (*bool, false*) – Switch between k-space (false) and imagespace based fitting (true).
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

run(*inp*, *data*, *iters*)

Optimization with 3D T(G)V regularization.

Parameters

- **(numpy.array)** (*x*) – Initial guess for the unknown parameters
- **(numpy.array)** – The complex valued data to fit.
- **iters** (*int*) – Number of primal-dual iterations to run

Returns A tupel of all primal variables (x,v in the Paper). If no streaming is used, the two entries are opf class PyOpenCL.Array, otherwise Numpy.Array.

Return type tupel

setFvalInit(*fval*)

Set the initial value of the cost function.

Parameters **fval** (*float*) – The initial cost of the optimization problem

updateRegPar(*irgn_par*)
Update the regularization parameters.

Performs an update of the regularization parameters as these usually vary from one to another Gauss-Newton step.

Parameters **(dic)** (*irgn_par*) –

update_Kyk2(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Precompute the v-part of the Adjoint Linear operator.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_primal(*outp*, *inp*, *par*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Primal update of the x variable in the Primal-Dual Algorithm.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_r(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Update the data dual variable r.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use

- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_v(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Primal update of the v variable in Primal-Dual Algorithm.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_z1(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Dual update of the z1 variable in Primal-Dual Algorithm for TGV.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_z1_tv(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Dual update of the z1 variable in Primal-Dual Algorithm for TV.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update

- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

update_z2(*outp*, *inp*, *par=None*, *idx=0*, *idxq=0*, *bound_cond=0*, *wait_for=None*)
Dual update of the z2 variable in Primal-Dual Algorithm for TGV.

Parameters

- **outp** (*PyOpenCL.Array*) – The result of the update step
- **inp** (*PyOpenCL.Array*) – The previous values of x
- **par** (*list*) – List of necessary parameters for the update
- **idx** (*int*) – Index of the device to use
- **idxq** (*int*) – Index of the queue to use
- **bound_cond** (*int*) – Apply boundary condition (1) or not (0).
- **wait_for** (*list of PyOpenCL.Events, None*) – A optional list for PyOpenCL.Events to wait for

Returns A PyOpenCL.Event to wait for.

Return type PyOpenCL.Event

class pyqmri.solver.**PDSolverStreamed**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *coils*, *model*, *imagespace=False*, ***kwargs*)
Streamed version of the PD Solver.

This class is the base class for the streamed array optimization.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted

- **imagespace** (*bool, false*) – Switch between imagespace (True) and k-space (false) based fitting.

unknown_shape
Size of the unknown array

Type tuple of int

model_deriv_shape
Size of the partial derivative array of the unknowns

Type tuple of int

grad_shape
Size of the finite difference based gradient

Type tuple of int

symgrad_shape
Size of the finite difference based symmetrized gradient. Defaults to None in TV based optimization.

Type tuple of int, None

data_shape
Size of the data to be fitted

Type tuple of int

data_trans_axes
Order of transpose of data axis, requried for streaming

Type list of int

data_shape_T
Size of transposed data.

Type tuple of int

class pyqmri.solver.**PDSolverStreamedTGV**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, *imagespace=False*, *SMS=False*, ***kwargs*)

Streamed TGV optimization.

This class performes streamd TGV optimization.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.

- **model** (*PyQMRI.Model*) – The model which should be fitted
- **imagespace** (*bool, false*) – Switch between imagespace (True) and k-space (false) based fitting.
- **SMS** (*bool, false*) – Switch between SMS (True) and standard (false) reconstruction.

alpha
alpha0 parameter for TGV regularization weight

Type float

beta
alpha1 parameter for TGV regularization weight

Type float

symgrad_shape
Size of the symmetrized gradient

Type tuple of int

class pyqmri.solver.**PDSolverStreamedTGVSMS**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, *imagespace=False*, ***kwargs*)

Streamed TGV optimization for SMS data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted
- **imagespace** (*bool, false*) – Switch between imagespace (True) and k-space (false) based fitting.
- **SMS** (*bool, false*) – Switch between SMS (True) and standard (false) reconstruction.

alpha
alpha0 parameter for TGV regularization weight

Type float

beta
alpha1 parameter for TGV regularization weight

Type float

symgrad_shape
Size of the symmetrized gradient

Type tuple of int

class pyqmri.solver.**PDSolverStreamedTV**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, *imagespace=False*, *SMS=False*, ***kwargs*)

Streamed TV optimization.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted
- **imagespace** (*bool, false*) – Switch between imagespace (True) and k-space (false) based fitting.
- **SMS** (*bool, false*) – Switch between SMS (True) and standard (false) reconstruction.

alpha
alpha0 parameter for TGV regularization weight

Type float

symgrad_shape
Size of the symmetrized gradient

Type tuple of int

class pyqmri.solver.**PDSolverStreamedTVSMS**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, *imagespace=False*, ***kwargs*)

Streamed TV optimization for SMS data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.

- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted
- **imagespace** (*bool, false*) – Switch between imagespace (True) and k-space (false) based fitting.

alpha
alpha0 parameter for TGV regularization weight

Type float

symgrad_shape
Size of the symmetrized gradient

Type tuple of int

class pyqmri.solver.**PDSolverTGV**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, ***kwargs*)

TGV Primal Dual splitting optimization.

This Class performs a primal-dual variable splitting based reconstruction on single precission complex input data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted

alpha
alpha0 parameter for TGV regularization weight

Type float

beta
alpha1 parameter for TGV regularization weight

Type float

class pyqmri.solver.**PDSolverTV**(*par*, *irgn_par*, *queue*, *tau*, *fval*, *prg*, *linop*, *coils*, *model*, ***kwargs*)

Primal Dual splitting optimization for TV.

This Class performs a primal-dual variable splitting based reconstruction on single precission complex input data.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **irgn_par** (*dict*) – A python dict containing the regularization parameters for a given gauss newton step.
- **queue** (*list of PyOpenCL.Queues*) – A list of PyOpenCL queues to perform the optimization.
- **tau** (*float*) – Estimated step size based on operator norm of regularization.
- **fval** (*float*) – Estimate of the initial cost function value to scale the displayed values.
- **prg** (*PyOpenCL.Program*) – A PyOpenCL Program containing the kernels for optimization.
- **linops** (*list of PyQMRI Operator*) – The linear operators used for fitting.
- **coils** (*PyOpenCL Buffer or empty list*) – The coils used for reconstruction.
- **model** (*PyQMRI.Model*) – The model which should be fitted

alpha

TV regularization weight

Type float

6.3 Streaming

Module holding the class for streaming operations on the GPU.

class pyqmri.streaming.**Stream**(*fun*, *outp_shape*, *inp_shape*, *par_slices*, *overlap*, *nslice*, *queue*, *num_dev*, *reverse=False*, *lhs=None*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Basic streaming Class.

This Class is responsible for performing asynchroneous transfer and computation on the GPU for arbitrary large numpy data.

Parameters **fun** (*list of functions*) –

This list contains all functions that should be executed on the GPU. The functions are executed in order from first to last element of the list.

outp_shape (list of tuple): The shape of the output array. Slice dimension is assumed to be the same as number of parallel slices plus overlap.

inp_shape (list of list of tuple): The shape of the input arrays. Slice dimension is assumed to be the same as number of parallel slices plus overlap.

par_slices [int] Number of slices computed in one transfer on the GPU.

overlap [int] Overlap of adjacent blocks

nslice [int] Total number of slices

queue [list of PyOpenCL.Queue] The OpenCL queues used for transfer and computation. 4 queues are used per device.

num_dev [int] Number of computation devices.

reverse [bool, false] Indicator of the streaming direction. If False, streaming will start at the first and end at the last slice. If True streaming will be performed vice versa

lhs [list of bool, None] Indicator for the norm calculation in the line search of TGV. lhs refers to left hand side. Needs to be passed if a norm should be computed.

DTYPE [numpy.dype, numpy.complex64] Complex data type.

fun
This list contains all functions that should be executed on the GPU. The functions are executed in order from first to last element of the list.

Type list of functions

num_dev
Number of computation devices.

Type int

slices
Number of slices computed in one transfer on the GPU.

Type int

overlap
Overlap of adjacent blocks

Type int

queue
The OpenCL queues used for transfer and computation. 4 queues are used per device.

Type list of PyOpenCL.Queue

reverse
Indicator of the streaming direction. If False, streaming will start at the first and end at the last slice. If True streaming will be performed vice versa

Type bool

NSlice
Total number of slices

Type int

num_fun
Total number of functions to stream (length of fun)

Type int

lhs
Indicator for the norm calculation in the line search of TGV. lhs refers to left hand side.

Type list of bool, None

at_end
Specifies if the end of the data slice dimension is reached

Type bool

inp (*list of list of list of PyOpenCL.Array*)
For each function a list of devices and a list of inputs is generated. E.g. for one function which needs two inputs and one computation device the list would have dimensions [1][1][2]

outp (*list of list of PyOpenCL.Array*)
For each function a list of devices with a single output is generated. E.g. for one function and one device the list would have dimension [1][1]

connectouttoin (*outpos*, *inpos*)
Connect output to input of functions within the object.

This function can be used to connect the output of a function to the input of another one used in the same stream object.

Parameters

- **outpos** (*int*) – The position in the list of outputs which should be connected to an input
- **(list of list of np.arrays** (*inpos*) – The position in the list of inputs which should be connected with an output

eval (*outp*, *inp*, *par=None*)
Evaluate all functions of the object.

Perform asynchroneous evaluation of the functions stored in fun.

Parameters

- **(list of np.arrays)** (*outp*) – Result of the computation for each function as numpy array
- **(list of list of np.arrays)** (*inp*) – For each function contains a list of numpy arrays used as input.
- **(list of list of parameters)** (*par*) – Optional list of parameters which should be passed to a function.

evalwithnorm (*outp*, *inp*, *par=None*)
Evaluate all functions of the object and returns norms.

Perform asynchroneous evaluation of the functions stored in fun. Same as eval but also computes the norm relevant for the linesearch in the TGV algorithm.

Parameters

- **outp** (*list of np.arrays*) – Result of the computation for each function as numpy array
- **(list of list of np.arrays)** (*inp*) – For each function contains a list of numpy arrays used as input.
- **par** (*list of list of parameters*) – Optional list of parameters which should be passed to a function.

Returns (lhs, rhs) The lhs and rhs for the linesearch used in the TGV algorithm.

Return type tuple of floats

6.4 Operators

Module holding the classes for different linear Operators.

class pyqmri.operator.**Operator**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Abstract base class for linear Operators used in the optimization.

This class serves as the base class for all linear operators used in the varous optimization algorithms. it requires to implement a forward and backward application in and out of place.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

NScan
Number of total measurements (Scans)

Type int

NC
Number of complex coils

Type int

NSlice
Number ofSlices

Type int

dimX
X dimension of the parameter maps

Type int

dimY
Y dimension of the parameter maps

Type int

N
N number of samples per readout

Type int

Nproj
Number of rreadouts

Type int

unknowns_TGV
Number of unknowns which should be regularized with TGV. It is assumed that these occure first in the unknown vector. Currently at least 1 TGV unknown is required.

Type int

unknowns_H1
Number of unknowns which should be regularized with H1. It is assumed that these occure after all TGV unknowns in the unknown vector. Currently this number can be zero which implies that no H1 regularization is used.

Type int

unknowns
The sum of TGV and H1 unknowns.

Type int

ctx
The context for the PyOpenCL computations. If streamed operations are used a list of ctx is required. One for each computation device.

Type list of PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels. If streamed operations are used a list of queues is required. Four for each computation device.

Type list of PyOpenCL.Queue

dz
The ratio between the physical X,Y dimensions vs the Z dimension. This allows for anisotrpic regularization along the Z dimension.

Type float

num_dev
Number of compute devices

Type int

NUFFT
A PyOpenCLnuFFT object to perform forward and backword transformations from image to k-space and vice versa.

Type PyQMRI.transforms.PyOpenCLnuFFT

prg
The PyOpenCL program containing all compiled kernels.

Type PyOpenCL.Program

self.**DTYPE**
Complex working precission. Currently single precission only.

Type numpy.dtype

self.**DTYPE_real**
Real working precission. Currently single precission only.

Type numpy.dtype

static GradientOperatorFactory(*par*, *prg*, *DTYPE*, *DTYPE_real*, *streamed=False*)
Gradient forward/adjoint operator factory method.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).

- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.
- **streamed** (*bool, false*) – Use standard reconstruction (false) or streaming of memory blocks to the compute device (true). Only use this if data does not fit in one block.

Returns A specialized instance of a PyQMRI.Operator to perform forward and ajoint gradient calculations.

Return type PyQMRI.Operator

static MRIOperatorFactory(*par*, *prg*, *DTYPE*, *DTYPE_real*, *trafo=False*, *imagespace=False*, *SMS=False*, *streamed=False*)

MRI forward/adjoint operator factory method.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.
- **trafo** (*bool, false*) – Select between radial (True) or cartesian FFT (false).
- **imagespace** (*bool, false*) – Select between fitting in imagespace (True) or k-space (false).
- **SMS** (*bool, false*) – Select between simulatneous multi-slice reconstruction or standard.
- **streamed** (*bool, false*) – Use standard reconstruction (false) or streaming of memory blocks to the compute device (true). Only use this if data does not fit in one block.

Returns

- *PyQMRI.Operator* – A specialized instance of a PyQMRI.Operator to perform forward and ajoint operations for fitting.
- *PyQMRI.NUFFT* – An instance of the used (nu-)FFT if k-space fitting is performed, None otherwise.

static SymGradientOperatorFactory(*par*, *prg*, *DTYPE*, *DTYPE_real*, *streamed=False*)

Symmetrized Gradient forward/adjoint operator factory method.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.

- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.
- **streamed** (*bool, false*) – Use standard reconstruction (false) or streaming of memory blocks to the compute device (true). Only use this if data does not fit in one block.

Returns A specialized instance of a PyQMRI.Operator to perform forward and ajoint symmetriced gradient calculations.

Return type PyQMRI.Operator

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorFiniteGradient**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Gradient operator.

This class implements the finite difference gradient operation and the adjoint (negative divergence).

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

ratio
Ratio between the different unknowns

Type list of PyOpenCL.Array

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

updateRatio(*inp*)

class pyqmri.operator.**OperatorFiniteGradientStreamed**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
Streamed gradient operator.

This class implements the finite difference gradient operation and the adjoint (negative divergence).

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

par_slices
Slices to parallel transfer to the compute device.

Type int

ratio
Ratio between the different unknowns

Type list of PyOpenCL.Array

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

getStreamedGradientObject()
Access privat stream gradient object.

Returns A PyQMRI streaming object for the gradient computation.

Return type PyqMRI.Streaming.Stream

updateRatio(*inp*)

class pyqmri.operator.**OperatorFiniteSymGradient**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
Symmetrized gradient operator.

This class implements the finite difference symmetrized gradient operation and the adjoint (negative symmetrized divergence).

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).

- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

ratio
Ratio between the different unknowns

Type list of PyOpenCL.Array

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)

Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

updateRatio(*inp*)

class pyqmri.operator.**OperatorFiniteSymGradientStreamed**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Streamed symmetrized gradient operator.

This class implements the finite difference symmetrized gradient operation and the adjoint (negative symmetrized divergence).

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

ctx

The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue

The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

par_slices
Slices to parallel transfer to the compute device.

Type int

ratio
Ratio between the different unknowns

Type list of PyOpenCL.Array

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

getStreamedSymGradientObject()
Access privat stream symmetrized gradient object.

Returns A PyQMRI streaming object for the symmetrized gradient computation.

Return type PyqMRI.Streaming.Stream

updateRatio(*inp*)

class pyqmri.operator.**OperatorImagespace**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
Imagespace based Operator.

This class serves as linear operator between parameter and imagespace.

Use this operator if you want to perform complex parameter fitting from complex image space data without the need of performing FFTs.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjKyk1(*out*, *inp*, ***kwargs*)

Apply the linear operator from image space to parameter space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex image space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)

Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)

Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)

Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorImagespaceStreamed**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

The streamed version of the Imagespace based Operator.

This class serves as linear operator between parameter and imagespace. All calculations are performed in a streamed fashion.

Use this operator if you want to perform complex parameter fitting from complex image space data without the need of performing FFTs. In contrast to non-streaming classes no out of place operations are implemented.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

overlap

Number of slices that overlap between adjacent blocks.

Type int

par_slices

Number of slices per streamed block

Type int

fwdstr

The streaming object to perform the forward evaluation

Type PyQMRI.Stream

adjstr

The streaming object to perform the adjoint evaluation

Type PyQMRI.Stream

adjstrKyk1
The streaming object to perform the adjoint evaluation including z1 of the algorithm.

Type PyQMRI.Stream

unknown_shape
Size of the parameter maps

Type tuple of int

data_shape
Size of the data

Type tuple of int

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjKyk1(*out*, *inp*)
Apply the linear operator from parameter space to image space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*numpy.Array*) – The complex parameter space data which is used as input.
- **inp** (*numpy.Array*) – The complex parameter space data which is used as input.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorKspace**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *trafo=True*)
k-Space based Operator.

This class serves as linear operator between parameter and k-space.

Use this operator if you want to perform complex parameter fitting from complex k-space data. The type of fft is defined through the NUFFT object. The NUFFT object can also be used for simple Cartesian FFTs.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.
- **trafo** (*bool, true*) – Switch between cartesian (false) and non-cartesian FFT (True, default).

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

NUFFT
The (nu) FFT used for fitting.

Type PyQMRI.PyOpenCLnuFFT

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjKyk1(*out*, *inp*, ***kwargs*)
Apply the linear operator from parameter space to k-space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorKspaceSMS**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
k-Space based Operator for SMS reconstruction.

This class serves as linear operator between parameter and k-space. It implements simultaneous-multi-slice (SMS) reconstruction.

Use this operator if you want to perform complex parameter fitting from complex k-space data measured with SMS. Currently only Cartesian FFTs are supported.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

packs
Number of SMS packs.

Type int

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

NUFFT
The (nu) FFT used for fitting.

Type PyQMRI.PyOpenCLnuFFT

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjKyk1(*out*, *inp*, ***kwargs*)
Apply the linear operator from parameter space to k-space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorKspaceSMSStreamed**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)
The streamed version of the k-space based SMS Operator.

This class serves as linear operator between parameter and k-space. It implements simultaneous-multi-slice (SMS) reconstruction.

All calculations are performed in a streamed fashion.

Use this operator if you want to perform complex parameter fitting from complex k-space data measured with SMS. Currently only Cartesian FFTs are supported.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.

- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

overlap
Number of slices that overlap between adjacent blocks.

Type int

par_slices
Number of slices per streamed block

Type int

packs
Number of packs to stream

Type int

fwdstr
The streaming object to perform the forward evaluation

Type PyQMRI.Stream

adjstr
The streaming object to perform the adjoint evaluation

Type PyQMRI.Stream

NUFFT
A list of NUFFT objects. One for each context.

Type list of PyQMRI.transforms.PyOpenCLnuFFT

FTstr
A streamed version of the used (non-uniform) FFT, applied forward.

Type PyQMRI.Stream

FTHstr
A streamed version of the used (non-uniform) FFT, applied adjoint.

Type PyQMRI.Stream

updateKyk1SMSstreamed

dat_trans_axes
Order in which the data needs to be transformed during the SMS reconstruction and streaming.

Type list of int

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array :param out: The complex parameter space data which is used as input. :type out: numpy.Array :param inp: The complex parameter space data which is used as input. :type inp: numpy.Array :param wait_for: A List of PyOpenCL events to wait for. :type wait_for: list of PyopenCL.Event

Returns The lhs and rhs for the line search of the primal-dual algorithm.

Return type tupel of floats

adjKyk1(*out*, *inp*, ***kwargs*)
Apply the linear operator from parameter space to k-space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*numpy.Array*) – The complex parameter space data which is used as input.
- **inp** (*numpy.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns The lhs and rhs for the line search of the primal-dual algorithm.

Return type tupel of floats

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)
Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

class pyqmri.operator.**OperatorKspaceStreamed**(*par*, *prg*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *trafo=True*)

The streamed version of the k-space based Operator.

This class serves as linear operator between parameter and k-space. All calculations are performed in a streamed fashion.

Use this operator if you want to perform complex parameter fitting from complex k-space data without the need of performing FFTs. In contrast to non-streaming classes no out of place operations are implemented.

Parameters

- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **prg** (*PyOpenCL.Program*) – The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.
- **trafo** (*bool, true*) – Switch between cartesian (false) and non-cartesian FFT (True, default).

overlap
Number of slices that overlap between adjacent blocks.

Type int

par_slices
Number of slices per streamed block.

Type int

fwdstr
The streaming object to perform the forward evaluation.

Type PyQMRI.Stream

adjstr
The streaming object to perform the adjoint evaluation.

Type PyQMRI.Stream

adjstrKyk1
The streaming object to perform the adjoint evaluation including z1 of the algorithm.

Type PyQMRI.Stream

NUFFT
A list of NUFFT objects. One for each context.

Type list of PyQMRI.transforms.PyOpenCLnuFFT

FTstr
A streamed version of the used (non-uniform) FFT, applied forward.

Type PyQMRI.Stream

unknown_shape
Size of the parameter maps

Type tuple of int

data_shape
Size of the data

Type tuple of int

adj(*out*, *inp*, ***kwargs*)
Adjoint operator application in-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex parameter space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex measurement space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

adjKyk1(*out*, *inp*)
Apply the linear operator from parameter space to k-space.

This method fully implements the combined linear operator consisting of the data part as well as the TGV regularization part.

Parameters

- **out** (*numpy.Array*) – The complex parameter space data which is used as input.
- **inp** (*numpy.Array*) – The complex parameter space data which is used as input.

adjoop(*inp*, ***kwargs*)
Adjoint operator application out-of-place.

Apply the linear operator from measurement space to parameter space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex measurement space which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

fwd(*out*, *inp*, ***kwargs*)
Forward operator application in-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array

Parameters

- **out** (*PyOpenCL.Array*) – The complex measurement space data which is the result of the computation.
- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns A PyOpenCL event to wait for.

Return type PyOpenCL.Event

fwdoop(*inp*, ***kwargs*)

Forward operator application out-of-place.

Apply the linear operator from parameter space to measurement space If streamed operations are used the PyOpenCL.Arrays are replaced by Numpy.Array This method need to generate a temporary array and will return it as the result.

Parameters

- **inp** (*PyOpenCL.Array*) – The complex parameter space data which is used as input.
- **wait_for** (*list of PyopenCL.Event*) – A List of PyOpenCL events to wait for.

Returns

- **PyOpenCL.Array** (*A PyOpenCL array containing the result of the*)
- *computation.*

6.5 Models

Package containig various model files for fitting.

This package contains the various MRI models currently implemented for the toolbox. In additon, the "GeneralModel" can be run with a simple text file to devine new model. An exemplary textfile for simple models can be generated by running the "genDefaultModelfile" function of the "GeneralModel".

Module holding the bi-exponential model for fitting.

class pyqmri.models.BiExpDecay.**Model**(*par*)

Bi-exponential model for MRI parameter quantification.

This class holds a bi-exponential model for fitting complex MRI data. It realizes a forward application of the analytical signal expression and the partial derivatives with respesct to each parameter of interest, as required by the abstract methods in the BaseModel.

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. TR, TE, TI, to fully describe the acquisitio process

TE

Echo time (or any other timing valid in a bi-exponential fit).

Type float

uk_scale

Scaling factors for each unknown to balance the partial derivatives.

Type list of float

guess
The initial guess. Needs to be set using "computeInitialGuess" prior to fitting.

Type numpy.array, None

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting.

Parameters **args** (*list of objects*) – Serves as universal interface. No objects need to be passed here.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor and applies a 1/x transformation for the time constants of the exponentials, yielding a result in milliseconds.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

Module holding the diffusion tensor model for fitting.

class pyqmri.models.DiffdirLL.**Model**(*par*)
Diffusion tensor model for MRI parameter quantification.

This class holds a DTI model for fitting complex MRI data. It realizes a forward application of the analytical signal expression and the partial derivatives with respesct to each parameter of interest, as required by the abstract methods in the BaseModel.

The fitting is based on the Cholesky decomposition of the DTI tensor to achiev an implicit positive definite constrained on each DTI tensor component.

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. TR, TE, TI, to fully describe the acquisitio process

b
b values for each diffusion direction.

Type float

dir
The diffusion direction vectors. Assumed to have length 1.

Type numpy.array

uk_scale
Scaling factors for each unknown to balance the partial derivatives.

Type list of float

guess
The initial guess. Needs to be set using "computeInitialGuess" prior to fitting.

Type numpy.array

phase
The phase of each diffusion direction relative to the b0 image. Estimated during the initial guess using the image series of all directions/bvalue pairs.

Type numpy.array

b0
The b0 image if present in the data file. None else.

Type numpy.array

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting. args[0] is assumed to contain the image series wich is used for phase correction.

Parameters **args** (*list of objects*) – Assumes the image series at potition 0 and optionally computes a phase based on the difference between each image series minus the first image in the series (Scan i minus Scan 0). This phase correction is needed as each diffusion weighting has a different phase.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor. As the DTI tensor is fitted using the Cholesky decompotion, each entry of the original tensor is recovered by combining the appropriate Cholesky factors after rescaling.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

Module holding the general model for fitting.

class pyqmri.models.GeneralModel.**Model**(*par*)
Realization of a generative model based on sympy.

This model can handel all kinds of sympy input in form of a config file. Partial derivatives of the model are automatically generated and a numpy compatible function is build from sumpy equations.

signaleq
The signal equation derived from sympy

Type sympy derived function

grad
Partial derivatives with respect to the unknowns

Type list of functions

rescalefun

Type list of functions

Functions to rescale each parameter

modelparams
List of model parameters

Type list

indphase
Flag to estimate the phase from a given image series. The phase is normed on the first image. If True, each image will be multiplied by the estimated phase in the forward and gradient evaluation.

Type bool

init_values
Initial guess for each unknown

Type list of str

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting, based on the values on the text file.

Parameters **args** (*list of objects*) – Assumes the image series at potition 0 and optionally computes a phase based on the difference between each image series minus the first image in the series. (Scan i minus Scan 0)

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor and an optional transformation.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

pyqmri.models.GeneralModel.**genDefaultModelfile**()
Generate a default model config file.

This method generates a default model file in the current project folder. This file can be modified or further models can be added.

Module holding the simple image model for image reconstruction.

class pyqmri.models.ImageReco.**Model**(*par*)
Image reconstruction model for MRI.

A simple linear image model to perform image reconstruction with joint regularization on all Scans.

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. TR, TE, TI, to fully describe the acquisitio process

guess
Initial guess for the images. Set after object creation using "computeInitialGuess"

Type numpy.array, None

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting.

Parameters **args** (*list of objects*) – Assumes the images series at position 0 and uses it as initial guess.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

Module holding the inversion recovers Look-Locker quantification model.

class pyqmri.models.IRLL.**Model**(*par*)

Inversion recovery Look-Locker model for MRI parameter quantification.

This class holds a IRLL model for T1 quantification from complex MRI data. It realizes a forward application of the analytical signal expression and the partial derivatives with respesct to each parameter of interest, as required by the abstract methods in the BaseModel. The fitting target is the exponential term itself which is easier to fit than the corresponding timing constant.

The rescale function applies a transformation and returns the expected T1 values in ms.

The implemented signal model follows the work from Henderson et al. (1999)

The model should only be used for radially acquired data!

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. TR, TE, TI, to fully describe the acquisitio process

TR

Repetition time of the IRLL sequence.

Type float

fa

Flip angle of the gradient echo sequence.

Type float

tau

Repetition time for one gradient echo read out.

Type float

td

Delay prio to first read-out point after inversion.

Type float

Nproj

Number of projections per bin

Type int

Nproj_measrued

Total number of projections measured

Type int

uk_scale

Scaling factors for each unknown to balance the partial derivatives.

Type list of float

guess

The initial guess. Needs to be set using "computeInitialGuess" prior to fitting.

Type numpy.array, None

scale

A scaling factor to balance the different exponential terms.

Type float

computeInitialGuess(**args*)

Initialize unknown array for the fitting.

This function provides an initial guess for the fitting.

Parameters **args** (*list of objects*) – Serves as universal interface. No objects need to be passed here.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor and applies a transformation for the time constants of the exponentials, yielding a resulting T1 in milliseconds.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

Module holding the template base class model.

class pyqmri.models.template.**BaseModel**(*par*)
Base model for MRI parameter quantification.

This class holds the base model to derive other signal models from. It defines abstract a forward application of the analytical signal expression and the partial derivatives with respesct to each parameter of interest.

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. Number of Scans, Slices, and image dimension.

constraints
An empy list of constrains objects.

Type list of pyqmri.models.template.constrains

NScan
Number of scans (dynamics).

Type int

NSlice
Number of slices.

Type int

dimX, dimY
The image dimensions.

Type int

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting.

execute_forward(*x*, *islice=None*)
Execute the signal model from parameter to imagespace.

This function exectues the given signal model to generate an image series, given estimated parameters.

Parameters

- **x** (*numpy.array*) – The array of quantitative parameters to be fitted
- **islice** (*int, None*) – Currently unused.

execute_gradient(*x*, *islice=None*)
Execute the partial derivatives of the signal model.

This function exectues the partial derivatives with respect to each unknown parameter, based on the signal model.

Parameters

- **x** (*numpy.array*) – The array of quantitative parameters to be fitted
- **islice** (*int, None*) – Currently unused.

plot_unknowns(*x*)
Plot the unkowns in an interactive figure.

This function can be used to plot intermediate results during the optimization process.

Parameters **x** (*dict*) – A Python dictionary containing the array of unknowns to be displayed, the associated names and real value constrains.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

class pyqmri.models.template.**constraints**(*min_val=-inf*, *max_val=inf*, *real_const=False*)
Constraints for a parameter.

This class holds min/max and real value constrains for a parameter. It supports updating these based on the current estimated sclaing between each partial derivative.

Parameters

- **min_val** (*float, -numpy.inf*) – The minimum value.
- **max_val** (*float, numpy.inf*) – The maximum value.
- **real_const** (*bool, false*) – Constrain to real values (true) or complex values (false).

constraints
An empy list of constrains objects.

Type list of pyqmri.models.template.constrains

NScan
Number of scans (dynamics).

Type int

NSlice
Number of slices.

Type int

dimX, dimY
The image dimensions.

Type int

figure
The placeholder figure object

Type matplotlib.pyplot.figure, None

update(*scale*)
Update the constrains based on current scaling factor.

Parameters **scale** (*float*) – The new scaling factor which should be used.

Module holding the variable flip angle model for T1 fitting.

class pyqmri.models.VFA.**Model**(*par*)
Variable flip angle model for MRI parameter quantification.

This class holds a variable flip angle model for T1 quantification from complex MRI data. It realizes a forward application of the analytical signal expression and the partial derivatives with respesct to each parameter of interest, as required by the abstract methods in the BaseModel. The fitting target is the exponential term itself which is easier to fit than the corresponding timing constant.

The rescale function applies a transformation and returns the expected T1 values in ms.

Parameters **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the sequence related parametrs, e.g. TR, TE, TI, to fully describe the acquisitio process

TR
Repetition time of the gradient echo sequence.

Type float

fa
A vector containing all flip angles, one per scan.

Type numpy.array

uk_scale
Scaling factors for each unknown to balance the partial derivatives.

Type list of float

guess
The initial guess. Needs to be set using "computeInitialGuess" prior to fitting.

Type numpy.array, None

computeInitialGuess(**args*)
Initialize unknown array for the fitting.

This function provides an initial guess for the fitting.

Parameters **args** (*list of objects*) – Serves as universal interface. No objects need to be passed here.

rescale(*x*)
Rescale the unknowns with the scaling factors.

Rescales each unknown with the corresponding scaling factor and applies a transformation for the time constants of the exponentials, yielding a resulting T1 in milliseconds.

Parameters **x** (*numpy.array*) – The array of unknowns to be rescaled

Returns The rescaled unknowns

Return type numpy.array

6.6 IRGN

Module holding the classes for IRGN Optimization without streaming.

class pyqmri.irgn.**IRGNOptimizer**(*par*, *model*, *trafo=1*, *imagespace=False*, *SMS=0*, *reg_type='TGV'*, *config=''*, *streamed=False*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*)

Main IRGN Optimization class.

This Class performs IRGN Optimization either with TGV or TV regularization.

Parameters

- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **model** (*pyqmri.model*) – Which model should be used for fitting. Expects a pyqmri.model instance.
- **trafo** (*int, 1*) – Select radial (1, default) or cartesian (0) sampling for the fft.
- **imagespace** (*bool, false*) – Perform the fitting from k-space data (false, default) or from the image series (true)
- **SMS** (*int, 0*) – Select if simultaneous multi-slice acquisition was used (1) or standard slice-by-slice acquisition was done (0, default).
- **reg_type** (*str, "TGV"*) – Select between "TGV" (default) or "TV" regularization.
- **config** (*str, ''*) – Name of config file. If empty, default config file will be generated.
- **streamed** (*bool, false*) – Select between standard reconstruction (false) or streamed reconstruction (true) for large volumetric data which does not fit on the GPU memory at once.
- **DTYPE** (*numpy.dtype, numpy.complex64*) – Complex working precission.
- **DTYPE_real** (*numpy.dtype, numpy.float32*) – Real working precission.

par

A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).

Type dict

gn_res

The residual values for of each Gauss-Newton step. Each iteration appends its value to the list.

Type list of floats

irgn_par

The parameters read from the config file to guide the IRGN optimization process

Type dict

execute(*data*)

Start the IRGN optimization.

This method performs iterative regularized Gauss-Newton optimization and calls the inner loop after precomputing the current linearization point. Results of the fitting process are saved after each linearization step to the output folder.

Parameters **data** (*numpy.array*) – the data to perform optimization/fitting on.

6.7 Transforms

Module holding the classes for different FFT operators.

class pyqmri.transforms.**PyOpenCL3DRadialNUFFT**(*ctx*, *queue*, *par*, *kwidth=5*, *klength=200*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *streamed=False*)

Non-uniform FFT object.

This class performs the 3D non-uniform FFT (NUFFT) operation. Linear interpolation of a sampled gridding kernel is used to regrid points from the non-cartesian grid back on the cartesian grid.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **kwidth** (*int*) – The width of the sampling kernel for regridding of non-uniform kspace samples.
- **klength** (*int*) – The length of the kernel lookup table which samples the contineous gridding kernel.
- **DTYPE** (*Numpy.dtype*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype*) – The real precision type. Currently float32 is used.

traj
The comlex sampling trajectory

Type PyOpenCL.Array

dcf
The densitiy compenation function

Type PyOpenCL.Array

ogf (*float*)
The overgriddingfactor for non-cartesian k-spaces.

fft_shape
3 dimensional tuple. Dim 0 containts all Scans, Coils and Slices. Dim 1 and 2 the overgridded image dimensions.

Type tuple of ints

fft_scale
The scaling factor to achieve a good adjointness of the forward and backward FFT.

Type float32

cl_kerneltable (*PyOpenCL.Buffer*)
The gridding lookup table as read only Buffer

cl_deapo (*PyOpenCL.Buffer*)
The deapodization lookup table as read only Buffer

par_fft
The number of parallel fft calls. Typically it iterates over the Scans.

Type int

fft
The fft object created from gpyfft (A wrapper for clFFT). The object is created only once an reused in each iterations, iterationg over all scans to keep the memory footprint low.

Type gpyfft.fft.FFT

prg
The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator. This will be determined by the factory and set after the object is created.

Type PyOpenCL.Program

FFT (*s*, *sg*, *wait_for=None*, *scan_offset=0*)
Perform the forward NUFFT operation.

Parameters

- **s** (*PyOpenCL.Array*) – The non-uniformly gridded k-space.
- **sg** (*PyOpenCL.Array*) – The complex image data.
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

FFTH (*sg*, *s*, *wait_for=None*, *scan_offset=0*)
Perform the inverse (adjoint) NUFFT operation.

Parameters

- **sg** (*PyOpenCL.Array*) – The complex image data.
- **s** (*PyOpenCL.Array*) – The non-uniformly gridded k-space
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

class pyqmri.transforms.**PyOpenCLCartNUFFT**(*ctx*, *queue*, *par*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *streamed=False*)

Cartesian FFT object.

This class performs the FFT operation.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **DTYPE** (*Numpy.dtype*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype*) – The real precision type. Currently float32 is used.

fft_shape
3 dimensional tuple. Dim 0 containts all Scans, Coils and Slices. Dim 1 and 2 the overgridded image dimensions.

Type tuple of ints

fft_scale
The scaling factor to achieve a good adjointness of the forward and backward FFT.

Type float32

par_fft
The number of parallel fft calls. Typically it iterates over the Scans.

Type int

fft
The fft object created from gpyfft (A wrapper for clFFT). The object is created only once an reused in each iterations, iterationg over all scans to keep the memory footprint low.

Type gpyfft.fft.FFT

mask
The undersampling mask for the Cartesian grid.

Type PyOpenCL.Array

prg
The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator. This will be determined by the factory and set after the object is created.

Type PyOpenCL.Program

FFT(*s*, *sg*, *wait_for=None*, *scan_offset=0*)
Perform the forward FFT operation.

Parameters

- **s** (*PyOpenCL.Array*) – The uniformly gridded k-space.
- **sg** (*PyOpenCL.Array*) – The complex image data.
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

FFTH(*sg*, *s*, *wait_for=None*, *scan_offset=0*)
Perform the inverse (adjoint) FFT operation.

Parameters

- **sg** (*PyOpenCL.Array*) – The complex image data.
- **s** (*PyOpenCL.Array*) – The uniformly gridded k-space
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns PyOpenCL.Event

Return type A PyOpenCL event to wait for.

class pyqmri.transforms.**PyOpenCLRadialNUFFT**(*ctx*, *queue*, *par*, *kwidth=5*, *klength=200*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *streamed=False*)
Non-uniform FFT object.

This class performs the non-uniform FFT (NUFFT) operation. Linear interpolation of a sampled gridding kernel is used to regrid points from the non-cartesian grid back on the cartesian grid.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **kwidth** (*int*) – The width of the sampling kernel for regridding of non-uniform kspace samples.
- **klength** (*int*) – The length of the kernel lookup table which samples the contineous gridding kernel.
- **DTYPE** (*Numpy.dtype*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype*) – The real precision type. Currently float32 is used.

traj
The comlex sampling trajectory

Type PyOpenCL.Array

dcf
The densitiy compenation function

Type PyOpenCL.Array

ogf (*float*)
The overgriddingfactor for non-cartesian k-spaces.

fft_shape
3 dimensional tuple. Dim 0 containts all Scans, Coils and Slices. Dim 1 and 2 the overgridded image dimensions.

Type tuple of ints

fft_scale
The scaling factor to achieve a good adjointness of the forward and backward FFT.

Type float32

cl_kerneltable (*PyOpenCL.Buffer*)
The gridding lookup table as read only Buffer

cl_deapo (*PyOpenCL.Buffer*)
The deapodization lookup table as read only Buffer

par_fft
The number of parallel fft calls. Typically it iterates over the Scans.

Type int

fft
The fft object created from gpyfft (A wrapper for clFFT). The object is created only once an reused in each iterations, iterationg over all scans to keep the memory footprint low.

Type gpyfft.fft.FFT

prg
The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator. This will be determined by the factory and set after the object is created.

Type PyOpenCL.Program

FFT (*s*, *sg*, *wait_for=None*, *scan_offset=0*)
Perform the forward NUFFT operation.

Parameters

- **s** (*PyOpenCL.Array*) – The non-uniformly gridded k-space.
- **sg** (*PyOpenCL.Array*) – The complex image data.
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

FFTH (*sg*, *s*, *wait_for=None*, *scan_offset=0*)
Perform the inverse (adjoint) NUFFT operation.

Parameters

- **sg** (*PyOpenCL.Array*) – The complex image data.
- **s** (*PyOpenCL.Array*) – The non-uniformly gridded k-space
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

class pyqmri.transforms.**PyOpenCLSMSNUFFT**(*ctx*, *queue*, *par*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *streamed=False*)

Cartesian FFT-SMS object.

This class performs the FFT operation assuming a SMS acquisition.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **par** (*dict A python dict containing the necessary information to*) – setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **DTYPE** (*Numpy.dtype*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype*) – The real precision type. Currently float32 is used.

fft_shape
3 dimensional tuple. Dim 0 containts all Scans, Coils and Slices. Dim 1 and 2 the overgridded image dimensions.

Type tuple of ints

fft_scale
The scaling factor to achieve a good adjointness of the forward and backward FFT.

Type float32

par_fft
The number of parallel fft calls. Typically it iterates over the Scans.

Type int

fft
The fft object created from gpyfft (A wrapper for clFFT). The object is created only once an reused in each iterations, iterationg over all scans to keep the memory footprint low.

Type gpyfft.fft.FFT

mask
The undersampling mask for the Cartesian grid.

Type PyOpenCL.Array

packs
The distance between the slices

Type int

MB
The multiband factor

Type int

shift
The vector pixel shifts used in the fft computation.

Type PyOpenCL.Array

prg
The PyOpenCL.Program object containing the necessary kernels to execute the linear Operator. This will be determined by the factory and set after the object is created.

Type PyOpenCL.Program

FFT(*s*, *sg*, *wait_for=None*, *scan_offset=0*)
Perform the forward FFT operation.

Parameters

- **s** (*PyOpenCL.Array*) – The uniformly gridded k-space compressed by the MB factor.
- **sg** (*PyOpenCL.Array*) – The complex image data.
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

FFTH(*sg*, *s*, *wait_for=None*, *scan_offset=0*)
Perform the inverse (adjoint) FFT operation.

Parameters

- **sg** (*PyOpenCL.Array*) – The complex image data.
- **s** (*PyOpenCL.Array*) – The uniformly gridded k-space compressed by the MB factor.
- **wait_for** (*list of PyopenCL.Event, None*) – A List of PyOpenCL events to wait for.
- **scan_offset** (*int, 0*) – Offset compared to the first acquired scan.

Returns **PyOpenCL.Event**

Return type A PyOpenCL event to wait for.

class pyqmri.transforms.**PyOpenCLnuFFT**(*ctx*, *queue*, *fft_dim*, *DTYPE*, *DTYPE_real*)
Base class for FFT calculation.

This class serves as the base class for all FFT object used in the varous optimization algorithms. It provides a factory method to generate a FFT object based on the input.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **fft_dim** (*tuple of int*) – The dimensions to take the fft over
- **DTYPE** (*Numpy.dtype*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype*) – The real precision type. Currently float32 is used.

DTYPE
The comlex precision type. Currently complex64 is used.

Type Numpy.dtype

DTYPE_real
The real precision type. Currently float32 is used.

Type Numpy.dtype

ctx
The context for the PyOpenCL computations.

Type PyOpenCL.Context

queue
The computation Queue for the PyOpenCL kernels.

Type PyOpenCL.Queue

prg
The PyOpenCL Program Object containing the compiled kernels.

Type PyOpenCL.Program

fft_dim
The dimensions to take the fft over

Type tuple of int

static create(*ctx*, *queue*, *par*, *kwidth=5*, *klength=1000*, *DTYPE=<class 'numpy.complex64'>*, *DTYPE_real=<class 'numpy.float32'>*, *radial=False*, *SMS=False*, *streamed=False*)
FFT factory method.

Based on the inputs this method decides which FFT object should be returned.

Parameters

- **ctx** (*PyOpenCL.Context*) – The context for the PyOpenCL computations.
- **queue** (*PyOpenCL.Queue*) – The computation Queue for the PyOpenCL kernels.
- **par** (*dict*) – A python dict containing the necessary information to setup the object. Needs to contain the number of slices (NSlice), number of scans (NScan), image dimensions (dimX, dimY), number of coils (NC), sampling points (N) and read outs (NProj) a PyOpenCL queue (queue) and the complex coil sensitivities (C).
- **kwidth** (*int, 5*) – The width of the sampling kernel for regridding of non-uniform kspace samples.
- **klength** (*int, 200*) – The length of the kernel lookup table which samples the contineous gridding kernel.
- **DTYPE** (*Numpy.dtype, numpy.complex64*) – The comlex precision type. Currently complex64 is used.
- **DTYPE_real** (*Numpy.dtype, numpy.float32*) – The real precision type. Currently float32 is used.
- **radial** (*bool, False*) – Switch for Cartesian (False) and non-Cartesian (True) FFT.
- **SMS** (*bool, False*) – Switch between Simultaneous Multi Slice reconstruction (True) and simple slice by slice reconstruction.
- **streamed** (*bool, False*) – Switch between normal reconstruction in one big block versus streamed reconstruction of smaller blocks.

Returns The setup FFT object.

Return type PyOpenCLnuFFT object

Raises AssertionError: – If the Combination of passed flags to choose the FFT aren't compatible with each other. E.g.: Radial and SMS True.

CHAPTER 7

Index

CHAPTER 8

Module Index

CHAPTER 9

Sample Data

In-vivo datasets used in the original publication (doi: [10.1002/mrm.27502]) can be found at zenodo. As these data sets are from an older release, the coil sensitivity profiles saved within the .h5 files need to be deleted prior to reconstruction. This invokes a new conputation of coil sensitivity profiles, matching the data within the fitting.

CHAPTER 10

Changelog:

v1.1 Added the first iteration of CPU support. Tested on Intel CPUS using pocl on Ubuntu and Arch Linux.

CHAPTER 11

Citation:

If using the toolbox, please consider citing: “Maier et al., (2020). PyQMRI: An accelerated Python based Quantitative MRI toolbox. Journal of Open Source Software, 5(56), 2727, https://doi.org/10.21105/joss.02727”

Also consider citing “Oliver Maier, Matthias Schloegl, Kristian Bredies, and Rudolf Stollberger; 3D Model-Based Parameter Quantification on Resource Constrained Hardware using Double-Buffering. Proceedings of the 27th meeting of the ISMRM, 2019, Montreal, Canada” if using parts of the software, specifically the PyOpenCL based NUFFT and the double buffering capabilities, in your work.

CHAPTER 12

Older Releases:

You can find the code for

Maier O, Schoormans J,Schloegl M, Strijkers GJ, Lesch A, Benkert T, Block T, Coolen BF, Bredies K, Stollberger R
Rapid T1 quantification from high resolution 3D data with model-based reconstruction.
Magn Reson Med., 2018; 00:1–16 doi: [10.1002/mrm.27502]

at [v0.1.0]

Python Module Index

p

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U