Project on Tomographic Reconstruction

In this project you will implement a regularized reconstruction method for 3D tomography in ODL[1] and apply it to a realistic 3D CT dataset.

Problem description

Let x be the image to be reconstructed, and let y be the result of the application of the ray transform (forward) operator A to x with additive noise e, that is, $y = \mathrm{A}(x) + e$. The goal is to estimate x using y. Since this problem is ill-posed, the typical approach is to add a regularization term. In this project you are going to use *Huber regularization*. This is a *Total Variation* type of regularization, where the ℓ_1-norm is replaced with the Huber norm H_ϵ (`odl.solvers.Huber`), defined in finite dimension as

$$\mathrm{H}_\epsilon(z) = \sum_i h_\epsilon(|z_i|) \tag{1}$$

for some $\epsilon > 0$, where $h_\epsilon \colon \mathbb{R} \to \mathbb{R}$ is the Huber function

$$h_\epsilon(t) = \begin{cases} \dfrac{t^2}{2\epsilon} & \text{if } |t| \leq \epsilon, \\ |t| - \dfrac{\epsilon}{2} & \text{otherwise.} \end{cases} \tag{2}$$

Note that for small ϵ, the Huber norm is a differentiable approximation of the ℓ_1-norm. Thus, instead of solving the least-squares problem

$$\min_x \frac{1}{2} \| \mathrm{A}(x) - y \|^2, \tag{3}$$

you must choose parameters $\lambda > 0$, $\epsilon > 0$, and solve the *regularized problem*

$$\min_x \frac{1}{2} \| \mathrm{A}(x) - y \|^2 + \lambda \, \mathrm{H}_\epsilon(\nabla x), \tag{4}$$

Here $\| \cdot \|$ is the norm on the *data space* (use `odl.solvers.L2Norm`) and ∇ is the gradient operator in the *image/reconstruction space* (use `odl.Gradient`). Since in practice images are discretized, this gradient operator is approximated by computing differences between adjacent pixels/voxels.

[1] https://odlgroup.github.io/odl/index.html

The idea of Huber regularization is to force adjacent pixels to have similar values by penalizing gradients. Specifically, the use of ℓ_1-norm (or Huber norm) promotes the type of solution where most of the gradients are close to 0, but some gradient values are allowed to be large. Hence, it promotes images with regions of near-constant values delineated by sharp borders, e.g., cartoon-like images. The choice of λ in (4) amounts to a trade-off between trying to find some x that fits the data, i.e., $\| \mathrm{A}(x) - y \|$ is small, without too much noisy variability, i.e., $\mathrm{H}_\epsilon(\nabla x)$ is small.

Note that an attempt to solve (3) directly leads to noisy pixel values (because of ill-posedness), and we saw in the labs that early stopping for iterative methods can yield reasonable solutions. Here, a suitable choice of the regularization parameter λ in (4) ensures that you do not need to use early stopping, since the issue of semi-convergence is (at least partially) taken care of by the regularization.

Dataset and problem parameters

The 3D CT dataset represents a realistic scan of a human head, obtained with the acquisition geometry of a real CT machine. More instructions on how to access the data and a server with GPU resources will be provided on the course page. For testing purposes in 2D (and 3D), use the Shepp-Logan phantom (`odl.phantom.shepp_logan`), noting that the contrast of the unmodified phantom (`modified=False`) is close to the true contrast of the CT dataset. While testing reconstruction methods in 2D, try to obtain enough contrast to distinguish at least some of the Shepp-Logan ellipsoids.

The 3D image to be reconstructed should have resolution $128 \times 128 \times 128$, with `min_pt` and `max_pt` set to $[-112.0, -112.0, 0.0]$ and $[112.0, 112.0, 224.0]$, respectively. To model the acquisition geometry, import `odl.contrib.tomo` and call `odl.contrib.tomo.elekta_icon_geometry` with `num_angles` set to 128. Combine this geometry with the reconstruction space to form the ray transform operator using `odl.tomo.RayTransform`. The 3D CT data should be in the range space of this operator.

To test reconstruction methods in 2D, the space should be a slice of the 3D space, i.e., with resolution 128×128 and `min_pt` resp. `max_pt` set to $[-112.0, -112.0]$ resp. $[112.0, 112.0]$. A suitable geometry is then `odl.tomo.cone_beam_geometry` with `src_radius` set to 780.0, `det_radius` set to 220.0, `num_angles` set to 128 and `short_scan` set to `True`.

Tasks

- Start by writing a script for solving the regularized problem in (4). You can use any solver implemented in ODL (`odl.solvers.smooth`), or implement your own solver. Note that functionals can be composed with operators using `odl.solvers.FunctionalComp`, and translated

by a vector using `odl.solvers.FunctionalTranslate`. Additional information about operator and functional arithmetic is available at `https://odlgroup.github.io/odl/guide/operator_guide.html`, `https://odlgroup.github.io/odl/guide/functional_guide.html`.

- Test your script by trying to reconstruct 2D images based on simulated noisy data using 2D phantoms (e.g., `odl.phantom.shepp_logan` with `modified=False`). For reference, the additive noise/corruption in the 3D CT data has a relative norm of about 1/100 (this ratio would otherwise have to be estimated from the data). If you implemented an algorithmic solver yourself, verify that it returns a result similar to that obtained with `odl.solvers.smooth.gradient.steepest_descent`.
- Based on your tests with the Shepp-Logan phantom in 2D, choose a parameter ϵ for the Huber norm and motivate your choice.
- For the regularization parameter λ, you should try to find a value that yields a good compromise between data-fit and regularization. Implement one of the methods from the lectures to find good regularization parameter values. Plot the 2D reconstruction error against the number of solver iterations for a few reasonable values of λ and noise levels.
- ONLY when the script is working for the 2D problem (and reasonable parameter values have been found) should you run it with the 3D CT data. This is important, since the running time for a solver can be very long, depending on the number of iterations. Start with a few iterations. Describe the effect of using different values for λ, and describe qualitatively the evolution of the reconstructed images as the number of iterations increases. Show some 2D-slices of the reconstructions.
- Compute a reconstruction using the Filtered Back-Projection method (`odl.tomo.fbp_op`). Vary the `filter_type` and `frequency_scaling` parameters. Frequency scaling is the relative frequency cut-off for the filter, so a smoother image can be obtained by choosing smaller values. Compare the results to those obtained in the previous tasks.
- Document your code and results in a written report.

Hints and code examples

- The following imports are recommended. The last entry provides access to the specific acquisition geometry for the data.

```
import numpy as np
import matplotlib.pyplot as plt
import odl
from odl.contrib import tomo
```

- The ray transform operator for 3D CT can be obtained as follows:

```
# Define the reconstruction space
reco_space = odl.uniform_discr(
    [-112, -112, 0],
    [112, 112, 224],
    (128, 128, 128),
)

# Define the acquisition geometry.
geometry = tomo.elekta_icon_geometry(
    num_angles=128,
)

# We can now create the ray transform operator.
# This operator takes elements in reco_space
# and returns elements in ray_trafo.range.
ray_trafo = odl.tomo.RayTransform(
    vol_space=reco_space,
    geometry=geometry,
)

# The data space is the range of the operator.
data_space = ray_trafo.range
```

- The ray transform operator for 2D tests can be defined as follows:

```
reco_space_2d = odl.uniform_discr(
    [-112, -112],
    [112, 112],
    (128, 128),
)
geometry_2d = odl.tomo.cone_beam_geometry(
    space=reco_space_2d,
    src_radius=780.0,
    det_radius=220.0,
    num_angles=128,
    short_scan=True,
)
ray_trafo_2d = odl.tomo.RayTransform(
    vol_space=reco_space_2d,
    geometry=geometry_2d,
)
data_space_2d = ray_trafo_2d.range
```

- The simplest way to minimize the objective function in (4) is to employ the steepest descent (gradient descent) method:

  1: x_0 arbitrary, $r_0 := -\nabla Q(x_0)$, t - step size.
  2: **for** $k := 0, 1, \ldots$ **do**
  3: $\quad x_{k+1} := x_k + t r_k$
  4: $\quad r_{k+1} := -\nabla Q(x_{k+1})$
  5: **end for**

  Note: Landweber is a special case of steepest descent.

- The above method may be improved by implementing a line search:

  1: x_0 arbitrary, $r_0 := -\nabla Q(x_0)$, t - step size.
  2: **for** $k := 0, 1, \ldots$ **do**
  3: $\quad x_{k+1} :=$ argmin of Q on the half-line $t \mapsto x_k + t r_k, \quad t \geq 0$
  4: $\quad r_{k+1} := -\nabla Q(x_{k+1})$
  5: **end for**

- Steepest descent converges if the step fulfills $0 < t < 2 * \|\nabla Q\|$.

- The operator ∇Q for a functional Q can be obtained with `Q.gradient`.

- An operator norm can be estimated with `.norm(estimate=True)`. If you have a composed operator you might want to estimate the norms of its components (`https://en.wikipedia.org/wiki/Operator_norm`).

- Structure the functions you write so that they can accept general input, e.g., a reconstruction method should work for both 2D and 3D spaces without issue.

- If your code is slow, try to avoid making copies of intermediate results.

- For more information about ODL, read the user's guide at `https://odlgroup.github.io/odl/guide/guide.html`.