CWI

Centrum Wiskunde & Informatica

ODL introduction

Holger Kohr
h.kohr@cwi.nl

Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Stockholm, December 12, 2017

Why a new software framework?

- Multiple modalities: CT, CBCT, PET, SPECT, spectral CT, phase contrast CT, electron tomography, ...
- Mathematical structures/notions: Functional, operator, Fréchet derivative, proximal, diffeomorphism, discretization, sparsifying transforms, ...
- Flexibility: Mathematical structures/notions re-usable across modalities
  $\rightsquigarrow$ Make it easy to "play around" with new ideas and combine concepts.
- Collaborative research: Need to share implementations of common concepts
- Reproducible research: Not enough to share theory and pseudocode, also need to share data and concrete implementations
  $\rightsquigarrow$ Software components need to be usable by others.

Conclusion: Need for a common software framework to exchange implementations of concepts and methods

Main components:

- Functional analysis module
  Handling of vector spaces, operators, discretizations – generally with a continuous point of view

Main components:

- Functional analysis module
  Handling of vector spaces, operators, discretizations – generally with a continuous point of view
- Optimization methods module
  General-purpose optimization methods suitable for solving inverse problems

Main components:

- Functional analysis module
  Handling of vector spaces, operators, discretizations – generally with a continuous point of view
- Optimization methods module
  General-purpose optimization methods suitable for solving inverse problems
- Tomography module
  Acquisition geometries and forward operators for tomographic applications

Main components:

- Library of atomic mathematical components
  - Deformation operators
  - Function transforms: wavelet, Fourier, ...
  - Differential operators: partial derivative, gradient, Laplacian, ...
  - Discretization-related: (re-)sampling, interpolation, domain extension, ...

Operator Discretization Library

A Python framework for inverse problems

Main components:

- Library of atomic mathematical components
  - Deformation operators
  - Function transforms: wavelet, Fourier, ...
  - Differential operators: partial derivative, gradient, Laplacian, ...
  - Discretization-related: (re-)sampling, interpolation, domain extension, ...
- Utility functions
  - Visualization: Slice viewer, real time plotting, ...
  - Phantoms: Shepp-Logan, FORBILD, Defrise, ...
  - Data I/O: MRC2014, Mayo Clinic, ...

Main components:

- User-contributed modules
  "Fast track" for experimental or slightly exotic code
  - Figures of Merit (FOMs) for image quality assessment
  - Handlers for specific data formats or geometries
  - Functionality to download and import public datasets
  - Wrappers for 3 major Deep Learning frameworks: **Tensorflow**, **Theano** and **Pytorch**

Design principle: modularity

Consider a TV minimization problem

$$\min_{f \in X} \left[\|A(f) - g\|_Y^2 + \lambda \mathrm{TV}(f) \right]$$

Design principle: modularity

Consider a TV minimization problem

$$\min_{f \in X} \left[\|A(f) - g\|_Y^2 + \lambda \mathrm{TV}(f) \right]$$

Components:

- *Reconstruction space* X
- *Data space* Y
- *Forward operator* $A \colon X \to Y$
- *Data* $g \in Y$
- *Data discrepancy functional* $\| \cdot - g\|_Y^2$
- *Regularization parameter* $\lambda > 0$
- *Regularization functional* $\mathrm{TV}(\cdot)$

Design principle: modularity

Consider a TV minimization problem

$$\min_{f \in X} \left[\|A(f) - g\|_Y^2 + \lambda \mathrm{TV}(f) \right]$$

Components:

- *Reconstruction space X*
- *Data space Y*
- *Forward operator $A \colon X \to Y$*
- *Data $g \in Y$*
- *Data discrepancy functional $\| \cdot - g\|_Y^2$*
- *Regularization parameter $\lambda > 0$*
- *Regularization functional $\mathrm{TV}(\cdot)$*

$\rightsquigarrow$ (almost) freely exchangeable "modules" in the mathematical formulation

$\rightsquigarrow$ ODL maps them to software objects as closely as possible

$\rightsquigarrow$ Mathematics as strong guideline for software design

$\rightsquigarrow$ Makes the software "feel" natural to mathematicians

Design principle: abstraction

Landweber's method: Determine f from given data $g = A(f)$ and an initial guess f_0 by

$$f_{k+1} = f_k + \omega A'(f_k)^* \big(g - A(f_k)\big), \quad k = 0, 1, \ldots, K - 1$$

Code using ODL:

```
def landweber(f, A, g, omega, K):
    for i in range(K):
        f += omega * A.derivative(f).adjoint(g - A(f))
```

Design principle: abstraction

Landweber's method: Determine f from given data $g = A(f)$ and an initial guess f_0 by

$$f_{k+1} = f_k + \omega A'(f_k)^* \big(g - A(f_k)\big), \quad k = 0, 1, \ldots, K-1$$

Code using ODL:

```
def landweber(f, A, g, omega, K):
    for i in range(K):
        f += omega * A.derivative(f).adjoint(g - A(f))
```

- Completely generic (expects operator, data, plus some parameters)
- Uses abstract properties of operators in the iteration:
  - `A(f)` $\longleftrightarrow A(f)$ (operator evaluation)
  - `A.derivative(f)` $\longleftrightarrow A'(f)$ (derivative *operator* at f)
  - `A.derivative(f).adjoint` $\longleftrightarrow A'(f)^*$ (adjoint of the derivative at f)

Design principle: abstraction

Landweber's method: Determine f from given data $g = A(f)$ and an initial guess f_0 by

$$f_{k+1} = f_k + \omega A'(f_k)^* \big(g - A(f_k)\big), \quad k = 0, 1, \ldots, K-1$$

Code using ODL:

```
def landweber(f, A, g, omega, K):
    for i in range(K):
        f += omega * A.derivative(f).adjoint(g - A(f))
```

- `A` is an `Operator` that implements a *generic, abstract* interface: `domain`, `range`, `derivative`, `adjoint`, operator *evaluation*
- Lots of tools to build complex operators from simple ones: operator arithmetic `A + B`, composition `A * B`, *product space* operators etc.
- There are *many* readily implement operators in ODL, all implementing the above interface

Further considerations

- ODL is a *prototyping* framework, not a black-box solution.
  - $\rightsquigarrow$ Give users freedom to experiment and tinker, do “unorthodox” things
  - $\rightsquigarrow$ Very little “intelligence” that guesses what a user wants
  - $\rightsquigarrow$ Instead: make things “just work” that a typical user would expect to work
- It should be fun to explore the “What if?” scenarios in existing examples.
- Make use of external highly optimized code for heavy tasks if adequate.
- Don’t sacrifice performance!
  - $\rightsquigarrow$ Use libraries in the most efficient way possible (avoid copies, operate in-place, work with low-dimensional arrays, vectorization, broadcasting, ...)
  - $\rightsquigarrow$ Compute on the GPU whenever possible (new fast back-end coming soon)

Example: tomography

Inverse Problem: Determine the attenuation coefficient $\mu\colon \mathbb{R}^2 \supset \Omega \to \mathbb{R}$ from its ray transform $R\colon L^2(\Omega) \to Y$ defined as

$$R\mu(\theta, s) = \int_{\mathbb{R}} \mu(t\theta + s\theta^{\perp})\mathrm{d}x, \quad \theta, \theta^{\perp} \in S^1, \langle \theta^{\perp}, \theta \rangle = 0$$

Example: tomography

Inverse Problem: Determine the attenuation coefficient $\mu\colon \mathbb{R}^2 \supset \Omega \to \mathbb{R}$ from its ray transform $R\colon L^2(\Omega) \to Y$ defined as

$$R\mu(\theta, s) = \int_{\mathbb{R}} \mu(t\theta + s\theta^{\perp})\mathrm{d}x, \quad \theta, \theta^{\perp} \in S^1, \langle \theta^{\perp}, \theta \rangle = 0$$

Given: Noisy data

$$g(\theta, s) \approx R\mu(\theta, s)$$

Regularization: Conjugate gradient (CGLS) with early termination

Example: Tomography

Implementation steps:

- Set up uniformly *discretized* image space $L^2(\Omega)$ with a rectangular domain Ω and $n_x \times n_y$ pixels
- Create parallel beam geometry with P angles and K detector pixels
- Define ray transform $R \colon L^2(\Omega) \to Y$ (the space Y is inferred from the geometry)
- Solve inverse problem using CGLS
- Display the results

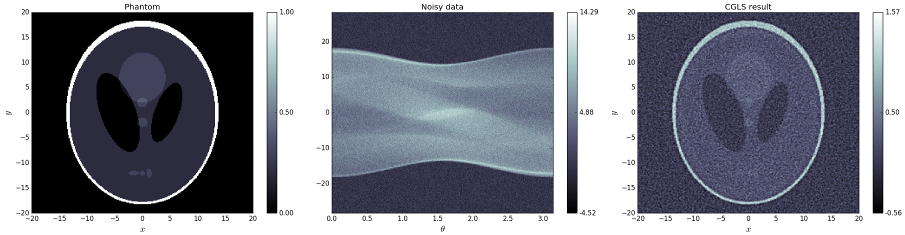
Example: Tomography

Code

```
# Create reconstruction space and ray transform
space = odl.uniform_discr([-20, -20], [20, 20], shape=(256, 256))
geometry = odl.tomo.parallel_beam_geometry(space, angles=1000)
ray_transform = odl.tomo.RayTransform(space, geometry)

# Create artificial data with around 5 % noise (data max = 10)
phantom = odl.phantom.shepp_logan(space, modified=True)
g = ray_transform(phantom)
g_noisy = g + 0.5 * odl.phantom.white_noise(ray_transform.range)

# Solve inverse problem
x = space.zero()
odl.solvers.conjugate_gradient_normal(ray_transform, x, g_noisy, niter=20)

# Display results
phantom.show('Phantom')
g_noisy.show('Noisy data')
x.show('CGLS after 20 iterations')
```

Example: Tomography

Results



Optimization example: preconditioned nonlinear ADMM

There is also a nonlinear version of ADMM [BKSV15] for the solution of problems of the form

$$\min_f \left[F(f) + G\big(L(f)\big)\right]$$

with convex functionals F and G and a possibly nonlinear operator L. It performs the following iteration, starting with $y^{(0)} = \mu^{(0)} = \bar{\mu}^{(0)} = 0$:

$$
\begin{aligned}
A^{(k)} &= \partial L(f^{(k)}) \\
\text{choose } \tau^{(k)} &< \big(\delta \|A^{(k)}\|^2\big)^{-1} \\
f^{(k+1)} &= \operatorname{prox}_{\tau^{(k)} F}\left[f^{(k)} - \tau^{(k)} \big(A^{(k)}\big)^* \bar{\mu}^{(k)}\right] \\
y^{(k+1)} &= \operatorname{prox}_{\sigma G}\left[y^{(k)} + \sigma\left(\mu^{(k)} + \delta\left(L(f^{(k+1)}) - y^{(k)}\right)\right)\right] \\
\mu^{(k+1)} &= \mu^{(k)} + \delta\left(L(f^{(k+1)}) - y^{(k+1)}\right) \\
\bar{\mu}^{(k+1)} &= 2\mu^{(k+1)} - \mu^{(k)}
\end{aligned}
$$

Optimization example: preconditioned nonlinear ADMM

Implementation

```
def admm_precon_nonlinear(x, f, g, L, delta, sigma, niter):
    y = L.range.zero()
    mu = delta * L(x)
    mubar = 2 * mu

    for i in range(niter):
        A = L.derivative(x)
        A_norm = power_method_opnorm(A)
        tau = 0.5 / (delta * A_norm ** 2)
        x[:] = f.proximal(tau)(x - tau * A.adjoint(mubar))
        y = g.proximal(sigma)((1 - sigma * delta) * y +
                              sigma * (mu + delta * L(x)))
        mu_old = mu
        mu = mu + delta * (L(x) - y)
        mubar = 2 * mu - mu_old
```

References

K. Bredies, K. Kunisch, and T. Pock.

Total Generalized Variation.
SIAM Journal on Imaging Sciences, 3(3):492–526, January 2010.

M. Benning, F. Knoll, C.-B. Schönlieb, and T. Valkonen.

Preconditioned ADMM with Nonlinear Operator Constraint.
In *System Modeling and Optimization*, IFIP Advances in Information and Communication Technology, pages 117–126. Springer, Cham, June 2015.

D. Cruz-Uribe and A. Fiorenza.

Variable Lebesgue Spaces: Foundations and Harmonic Analysis.
Springer Science & Business Media, 2013.

J. Duran, M. Moeller, C. Sbert, and D. Cremers.

Collaborative Total Variation: A General Framework for Vectorial TV Models.
SIAM Journal on Imaging Sciences, 9(1):116–151, January 2016.

M. J. Ehrhardt, K. Thielemans, L. Pizarro, D. Atkinson, Sébastien Ourselin, Brian F. Hutton, and Simon R. Arridge.

Joint reconstruction of PET-MRI by exploiting structural similarity.
Inverse Problems, 31(1):015001, 2015.

H. K.

Total variation regularization with variable Lebesgue prior.
arXiv:1702.08807 [math], February 2017.

N. Parikh and S. Boyd.

Proximal Algorithms.
Foundations and Trends in Optimization, 1(3):127–239, January 2014.

T. Wunderli.

On time flows of minimizers of general convex functionals of linear growth with variable exponent in BV space and stability of pseudosolutions.
Journal of Mathematical Analysis and Applications, 364(2):591–598, 2010.