Douglas—Rachford algorithm

The (Parallel) Douglas—Rachford ((P)DR) Algorithm was generalized to Hadamard manifolds in [BPS16].

The aim is to minimize the sum

$$F(p) = f(p) + g(p)$$

on a manifold, where the two summands have proximal maps $\operatorname{prox}_{\lambda f}, \operatorname{prox}_{\lambda g}$ that are easy to evaluate (maybe in closed form, or not too costly to approximate). Further, define the reflection operator at the proximal map as

$$\operatorname{refl}_{\lambda f}(p) = \operatorname{retr}_{\operatorname{prox}_{\lambda f}(p)}\bigl(-\operatorname{retr}^{-1}_{\operatorname{prox}_{\lambda f}(p)} p\bigr).$$

Let $\alpha_k \in [0,1]$ with $\sum_{k\in\mathbb{N}} \alpha_k(1-\alpha_k) = \infty$ and $\lambda > 0$ (which might depend on iteration k as well) be given.

Then the (P)DRA algorithm for initial data $x_0 \in \mathcal{H}$ as

Initialization

Initialize $q_0 = p_0$ and $k = 0$

Iteration

Repeat until a convergence criterion is reached

1. Compute $s_k = \operatorname{refl}_{\lambda f} \operatorname{refl}_{\lambda g}(q_k)$
2. Within that operation, store $p_{k+1} = \operatorname{prox}_{\lambda g}(t_k)$ which is the prox the inner reflection reflects at.
3. Compute $q_{k+1} = g(\alpha_k; q_k, s)$, where g is a curve approximating the shortest geodesic, provided by a retraction and its inverse
4. Set $k = k + 1$

until a stopping criterion is met.

Acceleration and Inertia

Before computing the first step, one can apply *inertia*: Given some $\theta_k \in (0,1)$ we can perform a step before, namely

$$t = \operatorname{retr}_{q_k}\bigl(-\theta_k \operatorname{retr}^{-1}_{q_k}(q_{k-1})\bigr),$$

that is adding inertia from the last two results computed in step 3 and use `t_k` as the argument of the double-reflection in step 1.

Instead of just computing step 4, one can also add an acceleration. Let `T` denote the double-reflection from the first step. Then, given a number n called *acceleration* step 3 is replaced with

$$q_{k+1} = T^n\bigl(g(\alpha_k; q_k, s)\bigr)$$

These both methods can also be combined by specifying inertia and acceleration

Result

The result is given by the last computed p_K.

For the parallel version, the first proximal map is a vectorial version where in each component one prox is applied to the corresponding copy of t_k and the second proximal map corresponds to the indicator function of the set, where all copies are equal (in $\mathcal{H}^n$, where n is the number of copies), leading to the second prox being the Riemannian mean.

Interface

`Manopt.DouglasRachford` — Function

```
DouglasRachford(M, f, proxes_f, p)
DouglasRachford(M, mpo, p)
```

Compute the Douglas-Rachford algorithm on the manifold $\mathcal{M}$, initial data p and the (two) proximal maps `proxMaps`, see Bergmann, Persch, Steidl, SIAM J Imag Sci, 2016.

For $k > 2$ proximal maps, the problem is reformulated using the parallel Douglas Rachford: A vectorial proximal map on the power manifold $\mathcal{M}^k$ is introduced as the first proximal map and the second proximal map of the is set to the `mean` (Riemannian Center of mass). This hence also boils down to two proximal maps, though each evaluates proximal maps in parallel, i.e.

component wise in a vector.

If you provide a `ManifoldProximalMapObjective` `mpo` instead, the proximal maps are kept unchanged.

Input

- `M` – a Riemannian Manifold $\mathcal M$
- `F` – a cost function consisting of a sum of cost functions
- `proxes_f` – functions of the form `(M, λ, p)->...` performing a proximal maps, where `λ` denotes the proximal parameter, for each of the summands of `F`. These can also be given in the `InplaceEvaluation` variants `(M, q, λ p) -> ...` computing in place of `q`.
- `p` – initial data $p \in \mathcal M$

Optional values

- `λ` – (`i-> 1.0`) function to provide the value for the proximal λ_i parameter during the calls
- `α` – (`i-> 0.9`) relaxation parameter α_i of the step from old to new iterate, $q^{(k+1)} = g(\alpha(k); q^{(k)}, s^{(k)})$, where $s^{(k)}$ is the result of the double
- `θ` – (`Nothing`) function (`i->0.0`) to provide interia θ_i. `Nothing` deactivates this and is equivalent to always returning 0.
- `inverse_retraction_method` – an inverse retraction method
- `n` – n-acceleration, apply the double reflection (T) `n` times after relaxation
- `R` – (`reflect` or `reflect!`) method employed in the iteration to perform the reflection of `x` at the prox `p`.
- `reflection_evaluation` – (`AllocatingEvaluation` `()`) specify whether the reflection works inplace or allocating (default)

whether `R` works inplace or allocating

- `retraction_method` – a retraction method
- `stop` – (`StopAfterIteration` `(300)`) a `StoppingCriterion`
- `parallel` – (`false`) indicate whether we are running a parallel Douglas-Rachford or not.
- `X` – (`zero_vector(M, p)`) a temporary storage for a tangent vector

and the ones that are passed to `decorate_state!` for decorators.

Output

the obtained (approximate) minimizer p^*, see `get_solver_return` for details

`Manopt.DouglasRachford!` — Function

```
DouglasRachford!(M, f, proxes_f, p)
DouglasRachford!(M, mpo, p)
```

Compute the Douglas-Rachford algorithm on the manifold $\mathcal M$, initial data $p \in \mathcal M$ and the (two) proximal maps `proxes_f` in place of `p`.

For $k > 2$ proximal maps, the problem is reformulated using the parallel Douglas Rachford: A vectorial proximal map on the power manifold $\mathcal M^k$ is introduced as the first proximal map and the second proximal map of the is set to the `mean` (Riemannian Center of mass). This hence also boils down to two proximal maps, though each evaluates proximal maps in parallel, i.e. component wise in a vector.

Note

While creating the new staring point `p'` on the power manifold, a copy of `p` Is created, so that the (by k>2 implicitly generated) parallel Douglas Rachford does not work in-place for now.

If you provide a `ManifoldProximalMapObjective` `mpo` instead, the proximal maps are kept unchanged.

Input

- `M` – a Riemannian Manifold $\mathcal M$
- `f` – a cost function consisting of a sum of cost functions
- `proxes_f` – functions of the form `(M, λ, p)->q` or `(M, q, λ, p)->q` performing a proximal map, where `λ` denotes the proximal parameter, for each of the summands of `f`.
- `p` – initial point $p \in \mathcal M$

For more options, see `DouglasRachford`.

State

`Manopt.DouglasRachfordState` — Type

```
DouglasRachfordState <: AbstractManoptSolverState
```

Store all options required for the DouglasRachford algorithm,

Fields

- `p` - the current iterate (result) For the (parallel) Douglas-Rachford. This is the “shadow sequence”, the proximal map of the first prox. For the parallel DR, this is a single point, not a point on the power manifold.
- `q` – (internal) the last result of the double reflection at the proxes relaxed. This is the variable converging to the fix point. Inertia and acceleration apply to this iterate.
- `q_old`, `q_tmp`, `p_base` – (internal) a temporary storage used within the double reflection or inertia.
- `λ` – (`i-> 1.0`) function to provide the value for the proximal λ_i parameter during the calls
- `α` – (`i-> 0.9`) relaxation parameter α_i of the step from old to new iterate, $q^{(k+1)} = g(\alpha(k); q^{(k)}, s^{(k)})$, where $s^{(k)}$ is the result of the double
- `θ` – (`Nothing`) function `(i->0.0)` to provide interia θ_i. `Nothing` deactivates this and is equivalent to always returning 0.
- `inverse_retraction_method` – an inverse retraction method
- `n` – n-acceleration, apply the double reflection (T) `n` times after relaxation
- `R` – (`reflect` or `reflect!`) method employed in the iteration to perform the reflection of `x` at the prox `p`.
- `reflection_evaluation` – (`AllocatingEvaluation()`) specify whether the reflection works inplace or allocating (default)

whether `R` works inplace or allocating

- `retraction_method` – a retraction method
- `stop` – (`StopAfterIteration(300)`) a `StoppingCriterion`
- `parallel` – (`false`) indicate whether we are running a parallel Douglas-Rachford or not.
- `X` – (`zero_vector(M, p)`) a temporary storage for a tangent vector

Constructor

```
DouglasRachfordState(M, p; kwargs...)
```

Generate the options for a Manifold `M` and an initial point `p`, where all fields with defaults above can be passed as keyword arguments.

For specific `DebugAction`s and `RecordAction`s see also Cyclic Proximal Point.

Furthermore, this solver has a short hand notation for the involved `reflect`ion.

`Manopt.reflect` — Function

```
reflect(M, f, x; kwargs...)
reflect!(M, q, f, x; kwargs...)
```

reflect the point `x` from the manifold `M` at the point `f(x)` of the function $f : \mathcal{M} \to \mathcal{M}$, i.e.,

$$\operatorname{refl}_f(x) = \operatorname{refl}_{f(x)}(x),$$

Compute the result in `q`.

see also `reflect` `(M,p,x)`, to which the keywords are also passed to.

```
reflect(M, p, x, kwargs...)
reflect!(M, q, p, x, kwargs...)
```

Reflect the point `x` from the manifold `M` at point `p`, i.e.

$$\operatorname{refl}_p(x) = \operatorname{retr}_p(-\operatorname{retr}^{-1}_p x).$$

where retr and retr^{-1} denote a retraction and an inverse retraction, respectively. This can also be done in place of `q`.

Keyword arguments

- `retraction_method` (`default_retraction_metiod(M, typeof(p))`) the retraction to use in the reflection
- `inverse_retraction_method` (`default_inverse_retraction_method(M, typeof(p))`) the inverse retraction to use within the reflection

and for the `reflect!` additionally

- `X` (`zero_vector(M,p)`) a temporary memory to compute the inverse retraction in place. otherwise this is the memory that would be allocated anyways.

Passing `X` to `reflect` will just have no effect.

Technical details

The `DouglasRachford` solver requires the following functions of a manifold to be available

- A `retract!(M, q, p, X)`; it is recommended to set the `default_retraction_method` to a favourite retraction. If this default is set, a `retraction_method=` does not have to be specified.
- An `inverse_retract!(M, X, p, q)`; it is recommended to set the `default_inverse_retraction_method` to a favourite retraction. If this default is set, a `inverse_retraction_method=` does not have to be specified.
- A `copyto!(M, q, p)` and `copy(M,p)` for points.

By default, one of the stopping criteria is `StopWhenChangeLess`, which requires

- An `inverse_retract!(M, X, p, q)`; it is recommended to set the `default_inverse_retraction_method` to a favourite retraction. If this default is set, a `inverse_retraction_method=` or `inverse_retraction_method_dual=` (for $\mathcal{N}$) does not have to be specified or the `distance(M, p, q)` for said default inverse retraction.

Literature