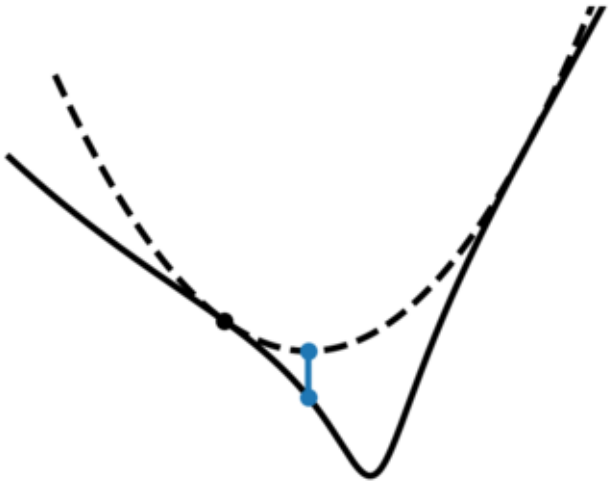


Q-MM Documentation

François Orieux

Jun 06, 2023

TABLE OF CONTENTS

1 Features **3**

2 Contribute **5**

3 Author **7**

4 Acknowledgement **9**

5 License **11**

5.1 Background 11
5.2 Example 12
5.3 References 12
5.4 Installation 13
5.5 Tutorial 13
5.6 API references (`qmm` module) 15
5.7 `operators` module 30
5.8 How to extend Q-MM 34

Bibliography **35**

Python Module Index **37**

Index **39**

Q-MM is a small Python toolbox to optimize differentiable objective functions

$$\hat{x} = \arg\min_{x \in \mathbb{R}} J(x)$$

by Majorization-Minimization with quadratic surrogate function. In particular, **no linesearch** is necessary and **close form formula for the step** are used with guaranteed convergence without sub-iteration. The explicit step formula allows fast convergence of the algorithm to a minimizer with minimal tuning parameters. However, the objective function must meet conditions, see *Background*.

CHAPTER

ONE

FEATURES

- The `mmmg` (or *3mg*), Majorize-Minimize Memory Gradient algorithm.
- The `mmcg`, Majorize-Minimize Conjugate Gradient algorithm.
- *No linesearch*: the step is obtained from a close form formula.
- *No conjugacy choice*: a conjugacy strategy is not necessary thanks to the subspace nature of the algorithms. The `mmcg` algorithm use a Polak-Ribière formula.
- Generic and flexible: there is no restriction on the number of regularizer, their type, .., as well as for data adequacy.
- Provided base class for objective allowing easier and fast implementation.
- Just one file if you like quick and dirty installation, but available with `pip`.
- Comes with examples of implemented linear operator.

CHAPTER

TWO

CONTRIBUTE

The code is hosted on Github under GPLv3 License. Feel free to contribute or submit issue.

CHAPTER

THREE

AUTHOR

If you are having issues, please let us know

orieux AT l2s.centralesupelec.fr

F. Orieux and R. Abirizk are affiliated to the Signal and Systems Laboratory L2S.

CHAPTER

FOUR

ACKNOWLEDGEMENT

Authors would like to thanks Jérôme Idier, Saïd Moussaoui and Émilie Chouzenoux. E. Chouzenoux has also a Matlab package that implements 3MG for image deconvolution on here webpage.

CHAPTER

FIVE

LICENSE

The project is licensed under the GPLv3 license and has a DOI with Zenodo. If you use the library, please cite it (you can change the version number).

```
@software{francois_orieux_2022_6373070,
  author       = {François Orieux and Ralph Abirizk},
  title        = {Q-MM: The Quadratic Majorize-Minimize Python
↪toolbox},
  month        = mar,
  year         = 2022,
  publisher    = {Zenodo},
  version      = {0.12.0},
  doi          = {10.5281/zenodo.6373069},
  url          = {https://doi.org/10.5281/zenodo.6373069}
}
```

and associated publications, see *Background*.

5.1 Background

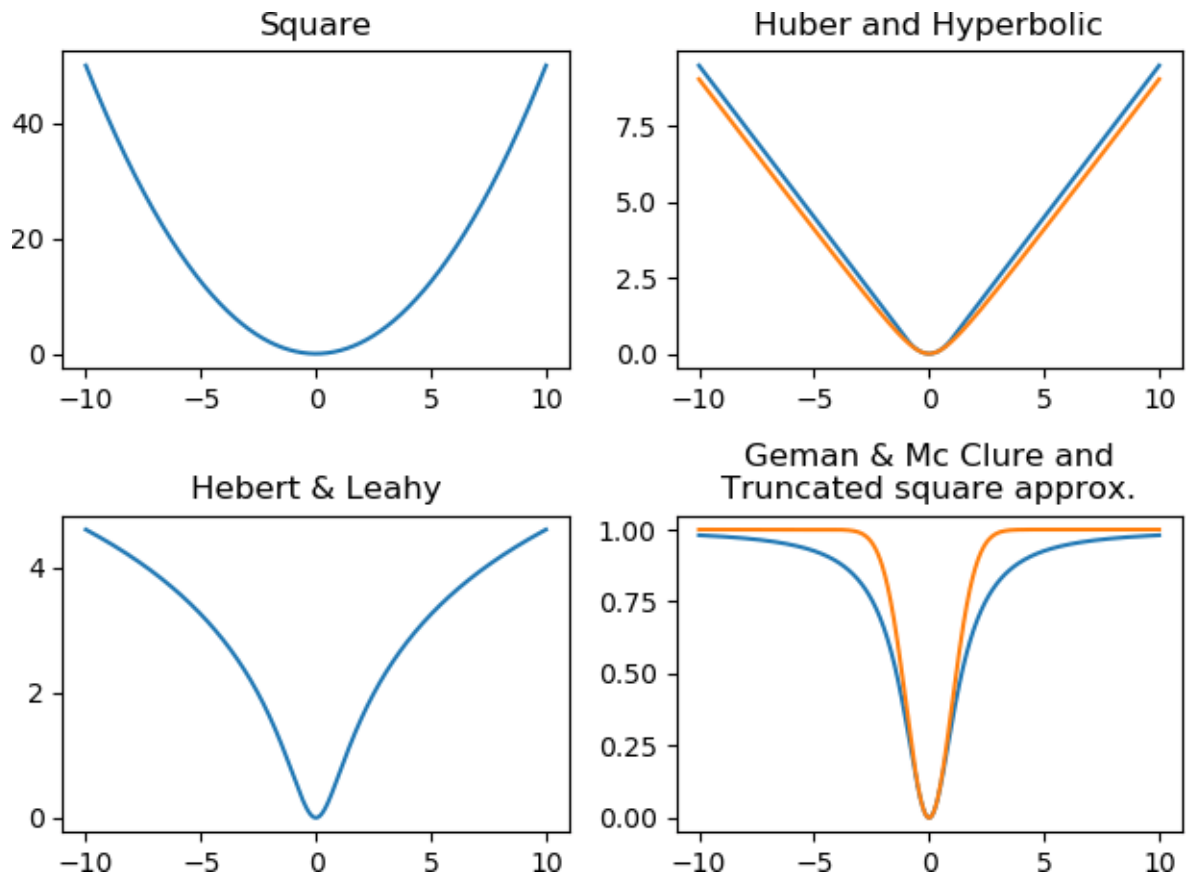
Q-MM is Python toolbox to optimise objective functions like

$$J(x) = \sum_k \mu_k \Psi_k(V_k x - \omega_k)$$

where x is the unknown of size N, V_k a matrix or linear operator of size $M_k \times N$, ω_k a data fixed vector of size M_k, μ_k a scalar hyperparameter, and $\Psi_k(u) = \sum_i \varphi_k(u_i)$. Q-MM suppose that the scalar functions φ are differentiable, even, coercive, $\varphi(\sqrt{\cdot})$ concave, and $0 < \dot{\varphi}(u)/u < +\infty$.

The optimization is done thanks to quadratic surrogate function. In particular, **no linesearch** is necessary and **close form formula for the step** are used with guaranteed convergence. The explicit step formula allows fast convergence of the algorithm to a minimizer of the objective function without tuning parameters. On the contrary, the objective must meet the conditions above.

The losses implemented in the toolbox, in addition to the square function, are illustrated below. The Geman & Mc Clure and the truncated square approximation are not coercive.



5.2 Example

A classical example is the resolution of an inverse problem with the minimization of

$$J(x) = \|y - Hx\|_2^2 + \mu\Psi(Vx)$$

where H is a low-pass forward model, V a regularization operator that approximate object gradient (kind of high-pass filter) and Ψ an edge preserving function like Huber. The above objective is obtained with $k \in \{1, 2\}$, $\Psi_1(\cdot) = \|\cdot\|_2^2$, $V_1 = H$, $\omega_1 = y$, and $\omega_2 = 0$.

5.3 References

The toolbox is funded on following papers. The papers [a], [b], and [c] are historical foundations. The implemented algorithms in the toolbox come from [d] and [e]. If you use this toolbox please cite these last two papers and this toolbox as

```
@software{qmm,
   title = {Q-MM: The Quadratic Majorize-Minimize Python toolbox},
   author = {Orieux, Fran\c{c}ois},
   url = {https://github.com/forieux/qmm},
}
```

5.4 Installation

Q-MM only depends on `numpy` and requires Python 3.6. The recommended way is to use poetry

```
poetry add qmm
```

but you can also use `pip` to install in system path

```
pip install qmm
```

or user's home

```
pip install --user qmm
```

Finally, since the toolbox is essentially just one file, and if `numpy` is installed, you can also just copy the `qmm` directory from Github realease (or by cloning) where your code can find it, or copy the `qmm.py` file and do, and do in both case

```
import qmm
```

5.5 Tutorial

The toolbox is just one file, the `qmm.py` module. The module contains essentially three part.

- The optimization algorithms, implemented as functions, that minimize

$$J(x) = \sum_k \mu_k \Psi_k(V_k x - \omega_k).$$

- The `Objective` class that implements

$$\mu\Psi(Vx - \omega) \quad \text{with} \quad \Psi(u) = \sum_i \varphi(u_i)$$

where u is a vector and φ a scalar function.

- The `Loss` class that implements φ.

5.5.1 Operators

The first thing to do is to implement the forward operator V and adjoint V^T. User is in charge of it. They are callable that could be Python functions or methods of objects.

```
def forward(array):
    # ...
    # do computation
    # ...
    return out   # An array or a list of array
```

(continues on next page)

(continued from previous page)

```
def adjoint(out):
    # ...
    # do computation
    # ...
    return array
```

The forward parameter must accept a `numpy.ndarray` x, of any shape, as unique parameter. I recommend using partial in the `functools` of the standard library, or using object oriented programming. The output of the forward operator must be

- a `ndarray` of any shape,
- or a list of `ndarray` (of any shape also).

Consequently, the adjoint operator must accept a `ndarray` or a list of `ndarray` as parameter, and returns a unique `ndarray`, of any shape, as output.

Note: The list of array allows mixed operators, like combination of data observation models, or multiple regularization.

Everything is internally vectorized. Therefore, the use of list of array implies memory copies of arrays.

5.5.2 Losses

The second step is to instantiate loss function φ, *`Huber`* for instance

```
from qmm import Huber, Objective, QuadObjective, mmmg
phi = Huber(delta=10)
```

Several losses are implemented, see *Background* and the *qmm* module.

5.5.3 Objectives

Then, a *`Objective`* $\mu\Psi(Vx)$ named `prior` can be instanced

```
prior = Objective(forward, adjoint, phi, hyper=0.01)
```

If a quadratic objective like $\|y - Hx\|_2^2$ is needed, the specific class *`QuadObjective`* can be used

```
data_adeq = QuadObjective(H, Ht, data=data)
```

Note: In the example above, the hyperparameter value is set to $\mu = 1$ and the data term is different that 0. For the prior term, the data is 0 by default and the hyperparameter is set to 0.01.

5.5.4 Optimization algorithms

Then you can run the algorithm, *mmmg()* for instance,

```
result = mmmg([data_adeq, prior], init, max_iter=200)
```

where the list `[data_adeq, prior]` means that the two objective functions are summed. The output *result* is an instance of *OptimizeResult*.

Note: *BaseObjective* can be summed returning a *MixedObjective* that behave like a list with additional functionalities. The above is equivalent to

```
fun = data_adeq + prior
result = mmmg(fun, init, max_iter=200)
```

Two algorithms are proposed :

- *mmcg()* that implements a Polak-Ribière Conjugate Gradient.
- *mmmg()* that implements a subspace by Memory-Gradient with 2D step (that, therefore, include the conjugacy parameter).

Both algorithms have close form formula for the 1D or 2D step by Majorization-Minimization Quadratic.

In addition a Linear Conjugate Gradient *lcg()* is implemented for quadratic objective.

5.6 API references (qmm module)

All the functionalities are provided by the unique `qmm` module described below.

5.6.1 Optimization algorithms

Three algorithms are implemented.

1. *mmcg()* that use the Majorize-Minimize Conjugate Gradient (MM-CG),
2. *mmmg()* that use the Majorize-Minimize Memory Gradient (3MG), and
3. *lcg()* that use the Linear Conjugate Gradient (CG) for quadratic objective *QuadObjective* only, with exact optimal step and conjugacy parameters.

The 3MG algorithm is usually faster but use more memory. The MM-CG can be faster and use less memory.

Majorize-Minimize Conjugate Gradient

mmcg(*objv_list*, *x0*, *tol=0.0001*, *max_iter=500*, *min_iter=0*, *precond=None*, *callback=None*, *calc_objv=False*)

The Majorize-Minimize Conjugate Gradient (MM-CG) algorithm.

The MM-CG is a nonlinear conjugate gradient (NL-CG) optimization algorithm with an explicit step formula based on Majorize-Minimize Quadratic approach[1].

Parameters

- **objv_list** (list of *BaseObjective*) – A list of `BaseObjective` objects that each represents a $\mu\ \psi(Vx - \omega)$. The objectives are summed.
- **x0** (*ndarray*) – The initial point.
- **tol** (*float, optional*) – The stopping tolerance. The algorithm is stopped when the gradient norm is inferior to *x0.size * tol*.
- **max_iter** (*int, optional*) – The maximum number of iterations.
- **min_iter** (*int, optional*) – The minimum number of iterations.
- **precond** (*callable, optional*) – A callable that must implement a preconditioner, that is Px. Must be a callable with a unique input parameter x and unique output like x.
- **callback** (*callable, optional*) – A function that receive the *OptimizeResult* at the end of each iteration.
- **calc_objv** (*boolean, optional*) – If True, objective function is computed at each iteration with low overhead. False by default. Not used by the algorithm.

Returns **result**

Return type *OptimizeResult*

[1] C. Labat and J. Idier, "Convergence of Conjugate Gradient Methods with a Closed-Form Stepsize Formula," J Optim Theory Appl, p. 18, 2008.

References

Majorize-Minimize Memory Gradient

mmmg(*objv_list*, *x0*, *tol=0.0001*, *max_iter=500*, *min_iter=0*, *precond=None*, *callback=None*, *calc_objv=False*)

The Majorize-Minimize Memory Gradient (*3MG*) algorithm.

The *mmmg* (*3MG*) algorithm is a subspace memory-gradient optimization algorithm with an explicit step formula based on Majorize-Minimize Quadratic approach[2].

Parameters

- **objv_list** (list of *BaseObjective*) – A list of `BaseObjective` objects that each represents a $\mu\ \psi(Vx - \omega)$. The objectives are summed.
- **x0** (*array*) – The initial point.
- **tol** (*float, optional*) – The stopping tolerance. The algorithm is stopped when the gradient norm is inferior to *x0.size * tol*.
- **max_iter** (*int, optional*) – The maximum number of iterations.
- **min_iter** (*int, optional*) – The minimum number of iterations.
- **precond** (*callable, optional*) – A callable that must implement a preconditioner, that is Px. Must be a callable with a unique input parameter x and unique output like x.
- **callback** (*callable, optional*) – A function that receive the *OptimizeResult* at the end of each iteration.
- **calc_objv** (*boolean, optional*) – If True, objective function is computed at each iteration with low overhead. False by default. Not used by the algorithm.

Returns result

Return type *OptimizeResult*

[2] E. Chouzenoux, J. Idier, and S. Moussaoui, "A Majorize-Minimize Strategy for Subspace Optimization Applied to Image Restoration," IEEE Trans. on Image Process., vol. 20, no. 6, pp. 1517–1528, Jun. 2011, doi: 10.1109/TIP.2010.2103083.

References

Linear Conjugate Gradient

lcg(*objv_list*, *x0*, *tol=0.0001*, *max_iter=500*, *min_iter=0*, *precond=None*, *callback=None*, *calc_objv=False*)
Linear Conjugate Gradient (CG) algorithm.

Linear Conjugate Gradient optimization algorithm for quadratic objective.

Parameters

- **objv_list** (list of *QuadObjective*) – A list of `QuadObjective` objects that each represents a ½ μ ||Vx - ω||^2_B. The objectives are summed.
- **x0** (*ndarray*) – The initial point.
- **precond** (*callable, optional*) – A callable that must implement a preconditioner, that is Px. Must be a callable with a unique input parameter x and unique output like x.
- **tol** (*float, optional*) – The stopping tolerance. The algorithm is stopped when the gradient norm is inferior to *x0.size* * *tol*.
- **max_iter** (*int, optional*) – The maximum number of iterations.
- **min_iter** (*int, optional*) – The minimum number of iterations.
- **callback** (*callable, optional*) – A function that receive the *OptimizeResult* at the end of each iteration.

Returns result

Return type *OptimizeResult*

5.6.2 Optimization results

The output are instance of `OptimizeResult` that behave like OptimizeResult of scipy. They behave like Python dictionary and are implemented to avoid dependency to scipy.

class OptimizeResult(**args*, ***kwargs*)
Represents the optimization result.

x: array The solution of the optimization, with same shape than *x0*.

success: bool Whether or not the optimizer exited successfully.

status: int Termination status of the optimizer. Its value depends on the underlying solver. Refer to message for details.

message: str Description of the cause of the termination.

nit: int Number of iterations performed by the optimizer.

diff: list of float The value of ||x_{k+1} - x_{k}||^2 at each iteration

time: list of float The time at each iteration, starting at 0, in seconds.

fun: float The value of the objective function.

objv_val: list of float The objective value at each iteration

jac: array The gradient of the objective function.

grad_norm: list of float The gradient norm at each iteration

Notes

`OptimizeResult` mimes *OptimizeResult* of scipy for compatibility.

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

5.6.3 Objective classes

Objective functions are defined from the abstract class `BaseObjective` that have three abstract methods that must be implemented by the subclass. If users want to implements it's own objective, he is encouraged to subclass `BaseObjective`.

Four generic concrete classes of `BaseObjective` can be used. The `Objective` class is the more general and prefered way, and `QuadObjective` is a specialized subclass that allows simplification and slightly faster computation. `Vmax` and `Vmin` are for bound penalties.

Note: The property `lastgv` is used by algorithms to compute the objective function value at each iteration with low overhead, if the flag `calc_fun` is set to `True` (`False` by default). It is not required by the algorithms.

---

class BaseObjective(*hyper=1*, *name=''*)
An abstract base class for objective function

$$J(x) = \mu\Psi\left(Vx - \omega\right)$$

with $\Psi(u) = \sum_i \varphi(u_i)$.

calc_objv
If true, compute the objective value when gradient is computed and store in *lastgv* attribute (False by default).

Type boolean

name
The name of the objective.

Type str

hyper
The hyperparameter value μ.

Type float

lastv
The last evaluated value of the objective (0 by default).

Type float

lastgv
The value of objective obtained during gradient computation (0 by default).

Type float

__init__ (*hyper=1*, *name=''*)
Initialize self. See help(type(self)) for accurate signature.

abstract operator (*point*)
Compute the output of *Vx*.

abstract value (*point*)
Compute the value at current point.

abstract gradient (*point*)
Compute the gradient at current point.

abstract norm_mat_major (*vecs*, *point*)
Return the normal matrix of the quadratic major function.

Given vectors *W = V·S*, return *W⊺·diag(b)·W*

where S are the vectors defining a subspace and *b* are Geman & Reynolds coefficients at given *point*.

Parameters

- **vecs** (*array*) – The *W* vectors.
- **point** (*array*) – The given point where to compute Geman & Reynolds coefficients *b*.

Returns **out** – The normal matrix

Return type array

Main objective

class Objective (*operator*, *adjoint*, *loss*, *data=None*, *hyper=1*, *name=''*)
An objective function defined as

$$J(x) = \mu\Psi\left(Vx - \omega\right)$$

with $\Psi(u) = \sum_i \varphi(u_i)$.

The instance attributs are:

data [array] The *data* array, or the vectorized list of array given at init.

hyper [float] The hyperparameter value μ.

loss [Loss] The loss φ.

__init__(*operator*, *adjoint*, *loss*, *data=None*, *hyper=1*, *name=''*)
A objective function $\mu\ \psi(Vx - \omega)$.

Parameters

- **operator** (*callable*) – A callable that compute the output Vx.
- **adjoint** (*callable*) – A callable that compute $V^t e$.
- **loss** (Loss) – The loss φ.
- **data** (*array or list of array, optional*) – The data vector ω.
- **hyper** (*float, optional*) – The hyperparameter μ.
- **name** (*str, optional*) – The name of the objective.

Notes

If *data* is a list of array, *operator* must return a similar list with arrays of same shape, and *adjoint* must accept a similar list also.

In that case, however, and for algorithm purpose, everything is internally stacked as a column vector and values are therefore copied, by using a *Stacked* object. This is not efficient but flexible. Users are encouraged to do the vectorization themselves and use this "list of array" feature.

operator(*point*)
Compute the output of Vx.

value(*point*)
The value of the objective function at given point

Return $\mu\ \psi(Vx - \omega)$.

gradient(*point*)
The gradient and value at given point

Return $\mu\ V^t \varphi'(Vx - \omega)$.

norm_mat_major(*vecs*, *point*)
Return the normal matrix of the quadratic major function.

Given vectors $W = V \cdot S$, return $W^t \cdot diag(b) \cdot W$

where S are the vectors defining a subspace and b are Geman & Reynolds coefficients at given *point*.

Parameters

- **vecs** (*array*) – The W vectors.

- **point** (*array*) – The given point where to compute Geman & Reynolds coefficients *b*.

Returns **out** – The normal matrix

Return type array

gr_coeffs (*point*)
The Geman & Reynolds coefficients at given point

Given *x* return *φ'(Vx - ω) / (Vx - ω)*.

gy_coeffs (*point*)
The Geman & Yang coefficients at given point

Given *x* return *Vx - φ'(Vx - ω)*.

Note: The *Objective* class implements `__call__` interface allowing objects to behave like callable (function), returning the objective value

```
identity = lambda x: x

objv = qmm.Objective(identity, identity, qmm.Square())
x = np.random.standard_normal((100, ))
objv(x) == objv.value(x)
```

Quadratic objective

This class implements specific properties or methods associated to quadratic objective function.

class QuadObjective (*operator*, *adjoint*, *hessp=None*, *data=None*, *hyper=1*, *invcovp=None*, *name=''*)
A quadratic objective function

$$\begin{aligned} J(x) &= \frac{1}{2}\mu\|Vx-\omega\|_B^2 \\ &= \frac{1}{2}\mu(Vx-\omega)^t B(Vx-\omega) \end{aligned} \tag{5.1}$$

The instance attributes are:

hyper [float] The hyperparameter value μ.

ht_data [array] The retroprojected data μ *V□Bω*.

constant [float] The constant value μ *ω□Bω*.

__init__ (*operator*, *adjoint*, *hessp=None*, *data=None*, *hyper=1*, *invcovp=None*, *name=''*)
A quadratic objective *½μ ||Vx - ω||²_B*

Parameters

- **operator** (*callable*) – A callable that compute the output *Vx*.

- **adjoint** (*callable*) – A callable that compute $V^T e$.
- **hessp** (*callable, optional*) – A callable that compute Qx as $Qx = V^T B V x$. Must take a parameter like *operator* and return a parameter like *adjoint* (x-like in both case).
- **data** (*array or list of array, optional*) – The data vector ω.
- **hyper** (*float, optional*) – The hyperparameter μ.
- **invcovp** (*callable, optional*) – A callable, that take a parameter like *adjoint* and return like *operator* (ω-like in both case), that apply the inverse covariance, or metric, $B=\Sigma^{-1}$. Equivalent to Identity if *None*.
- **name** (*str, optional*) – The name of the objective.

Notes

The *hessp* (Q) callable is used for gradient computation as $\nabla = \mu\ (Qx - b)$ where $b = V^T B \omega$ instead of $\nabla = \mu\ V^T B (Vx - \omega)$. This is optional and in some case can be more efficient. Use it only in that case.

The variable $b = V^T B \omega$ is computed at object initialisation.

property VtB_data
The second term $b = \mu \sum_i V_i^T B_i \omega_i$

property constant
The constant $c = \mu \sum\_i \omega_i^T B_i \omega_i$

operator (*point*)
Compute the output of Vx.

value (*point*)
The value of the objective function at given point

Return $\frac{1}{2} \mu\ ||Vx - \omega||^2\_B$.

gradient (*point*)
The gradient and value at given point

Return $\nabla = \mu\ (Qx - b) = \mu\ V^T B (Vx - \omega)$.

Notes

Objective value is always computed with low overhead thanks to the relation

$J(x) = ½ (x·∇ - x·b + \mu\ \omega·B\omega)$.

value_hessp (*point*, *hessp*)
Return $J(x)$ value at low cost given x and $q = Qx$

thanks to the relation

$J(x) = ½ (x·q - 2\ x·b + \mu\ \omega·B\omega)$.

value_residual (*point*, *residual*)
Return $J(x)$ value at low cost given x and $r = b - Qx$

thanks to the relation

$J(x) = ½ (x·(-b - r) + \mu\ \omega·B\omega)$.

norm_mat_major (*vecs*, *point*)
Return the normal matrix of the quadratic major function.

Given vectors $W = V·S$, return $W·diag(b)·W$

where S are the vectors defining a subspace and b are Geman & Reynolds coefficients at given *point*.

Parameters

- **vecs** (*array*) – The W vectors.
- **point** (*array*) – The given point where to compute Geman & Reynolds coefficients b.

Returns **out** – The normal matrix

Return type array

Note: The `operator` argument for *Objective* and *QuadObjective* must be a callable that accept an `array` as input. The operator can return an array as output but **can also return** a `list` of array (for data fusion for instance). However, for needs of optimization algorithm implementation, everything must be an array internally. In case of `list` or arrays, all these arrays are handled by a *Stacked* class, internally vectorized and the data are therefore memory copied, at each iteration.

If `operator` returns a list of array, the `adjoint` **must** accept a list of array also. Again, everything is vectorized and *Objective* rebuild the list of array internally.

QuadObjective handle this `list` of array more efficiently since data is not stored internally by the class but only $\mu V^T B\omega$, that is an array like x.

If given, the `hessp` callable argument for *QuadObjective* must accept an array and returns an array.

Specific objective classes

class Vmin (*vmin*, *hyper*, *name=''*)
A minimum value objective function

$$J(x) = \frac{1}{2}\mu \| P_{]-\infty,m]}(x) - m \|_2^2.$$

vmin [float] The minimum value m.

hyper [float] The hyperparameter value μ.

__init__ (*vmin*, *hyper*, *name=''*)
A minimum value objective function

J(x) = ½μ ||P_[m, +∞[(x) - m||².

Parameters

- **vmin** (*float*) – The minimum value m.
- **hyper** (*float*) – The hyperparameter value μ.
- **name** (*str*) – The name of the objective.

operator (*point*)
Compute the output of *Vx*.

value (*point*)
Return the value at current point.

gradient (*point*)
Compute the gradient at current point.

norm_mat_major (*vecs*, *point*)
Return the normal matrix of the quadratic major function.

Given vectors *W = V·S*, return *W□·diag(b)·W*

where S are the vectors defining a subspace and *b* are Geman & Reynolds coefficients at given *point*.

Parameters

- **vecs** (*array*) – The *W* vectors.
- **point** (*array*) – The given point where to compute Geman & Reynolds coefficients *b*.

Returns **out** – The normal matrix

Return type array

class Vmax (*vmax*, *hyper*, *name=''*)
A maximum value objective function

$$J(x) = \frac{1}{2}\mu \| P_{[M,+\infty[}(x) - M \|_2^2.$$

vmax [float] The maximum value M.

hyper [float] The hyperparameter value μ.

__init__ (*vmax*, *hyper*, *name=''*)
A maximum value objective function

Return $J(x) = ½\mu\ ||P\_[M, +\infty[(x) - M||^2$.

Parameters

- **vmax** (*float*) – The maximum value M.
- **hyper** (*float*) – The hyperparameter value μ.
- **name** (*str*) – The name of the objective.

operator (*point*)
Compute the output of Vx.

value (*point*)
Return the value at current point.

gradient (*point*)
Compute the gradient at current point.

norm_mat_major (*vecs*, *point*)
Return the normal matrix of the quadratic major function.

Given vectors $W = V \cdot S$, return $W? \cdot diag(b) \cdot W$

where S are the vectors defining a subspace and b are Geman & Reynolds coefficients at given *point*.

Parameters

- **vecs** (*array*) – The W vectors.
- **point** (*array*) – The given point where to compute Geman & Reynolds coefficients b.

Returns **out** – The normal matrix

Return type array

Sum of objectives

The *MixedObjective* is a convenient (not required) list-like class that represent the sum of *BaseObjective*. Moreover, *BaseObjective* and *MixedObjective* support the "+" operator and returns a *MixedObjective* instance, or update the instance, respectively. Since *MixedObjective* is a list, it can be used with *optimization algorithms*.

```
likelihood = QuadObjective(...)
prior1 = Objective(...)
prior2 = Objective(...)

# Equivalent to objective = MixedObjective([likelihood, prior1])
objective = likelihood + prior1
```

(continues on next page)

(continued from previous page)

```
# Equivalent to objective.append(prior2)
objective = objective + prior2

# Equivalent to res = mmmg([likelihood, prior1, prior2], ...)
res = mmmg(objective, ...)
```

class MixedObjective(*objv_list*)

Represents a mixed objective function

$$J(x) = \sum_k \mu_k \Psi_k \left(V_k x - \omega_k\right)$$

This is a *Sequence* (or list-like) and instance of this class can be used in optimization algorithms.

__init__(*objv_list*)

A mixed objective function

J(x) = ∑ᵢ μᵢ ψᵢ(Vᵢ·x - ωᵢ).

Parameters **objv_list** (list of *BaseObjective*) –

property lastv

Return the value of objectives obtained during gradient computation.

insert(*index*, *value*)

S.insert(index, value) – insert value before index

value(*point*)

The value J(x)

gradient(*point*)

The gradient ∇J(x)

5.6.4 Losses classes

The class *Loss* is an abstract base class and serve as parent class for all losses. At that time, the provided concrete loss functions are *Square*, *Huber*, *Hyperbolic*, *HebertLeahy*, *GemanMcClure*, and *TruncSquareApprox*.

Note: The *Loss* class implements `__call__` interface allowing objects to behave like callable (function), returning the function value

```
u = np.linspace(-5, 5, 1000)
pot = qmm.Huber(1)
plt.plot(u, pot(u))
```

---

class Loss(*inf*, *convex=False*, *coercive=False*)

An abstract base class for loss φ.

The class has the following attributes.

inf [float] The value of *lim_{u→0} φ'(u) / u*.

convex [boolean] A flag indicating if the loss is convex (not used).

coercive [boolean] A flag indicating if the loss is coercive (not used).

__init__(*inf*, *convex=False*, *coercive=False*)
The loss φ

Parameters

- **inf** (*float*) – The value of *lim_{u→0} φ'(u) / u*.
- **convex** (*boolean*) – A flag indicating if the loss is convex.
- **coercive** (*boolean*) – A flag indicating if the loss is coercive.

abstract value(*point*)
The value *φ(·)* at given point.

abstract gradient(*point*)
The gradient *φ'(·)* at given point.

gr_coeffs(*point*)
The Geman & Reynolds *φ'(·)/·* coefficients at given point.

gy_coeffs(*point*)
The Geman & Yang *· - φ'(·)* coefficients at given point.

Square

class Square
The Square loss

$$\varphi(u) = \frac{1}{2}u^2.$$

__init__()
The Square loss *φ(u) = ½ u²*.

value(*point*)
The value *φ(·)* at given point.

gradient(*point*)
The gradient *φ'(·)* at given point.

Huber

class Huber(*delta*)

The convex coercive Huber loss

$$\varphi(u) = \begin{cases} \frac{1}{2}u^2 & \text{, if } u \leq \delta, \\ \delta|u| - \frac{\delta^2}{2} & \text{, otherwise.} \end{cases}$$

__init__(*delta*)

The Huber loss.

value(*point*)

The value $\varphi(\cdot)$ at given point.

gradient(*point*)

The gradient $\varphi'(\cdot)$ at given point.

Hyperbolic or Pseudo-Huber

class Hyperbolic(*delta*)

The convex coercive hyperbolic loss

$$\varphi(u) = \delta^2 \left(\sqrt{1 + \frac{u^2}{\delta^2}} - 1 \right)$$

This is sometimes called Pseudo-Huber.

__init__(*delta*)

The hyperbolic loss.

value(*point*)

The value $\varphi(\cdot)$ at given point.

gradient(*point*)

The gradient $\varphi'(\cdot)$ at given point.

Hebert & Leahy

class HebertLeahy(*delta*)

The non-convex coercive Hebert & Leahy loss

$$\varphi(u) = \log \left(1 + \frac{u^2}{\delta^2} \right)$$

__init__(*delta*)

The Hebert & Leahy loss.

value(*point*)

The value $\varphi(\cdot)$ at given point.

gradient (*point*)
The gradient $\varphi'(\cdot)$ at given point.

Geman & Mc Clure

class GemanMcClure (*delta*)
The non-convex non-coervice Geman & Mc Clure loss

$$\varphi(u) = \frac{u^2}{2\delta^2 + u^2}$$

__init__ (*delta*)
The Geman & Mc Clure loss.

value (*point*)
The value $\varphi(\cdot)$ at given point.

gradient (*point*)
The gradient $\varphi'(\cdot)$ at given point.

Truncated Square approximation

class TruncSquareApprox (*delta*)
The non-convex non-coercive truncated square approximation

$$\varphi(u) = 1 - \exp\left(-\frac{u^2}{2\delta^2}\right)$$

__init__ (*delta*)
The truncated square approximation.

value (*point*)
The value $\varphi(\cdot)$ at given point.

gradient (*point*)
The gradient $\varphi'(\cdot)$ at given point.

5.7 operators module

Implements high level interface to manipulate linear operators. This module is not required by Q-MM, is basic, but can serve as guide or for reuse.

dft2 (*obj*)
Return the orthogonal real 2D fft.

Parameters **obj** (*array-like*) – The array on which to perform the 2D DFT.

Returns **out**

Return type array

Notes

This function is a wrapper of numpy.fft.rfft2. The DFT is made on the two last axis.

idft2 (*obj*, *shape*)
Return the orthogonal real 2D ifft.

Parameters

- **obj** (*array-like*) – The array on which to perform the inverse 2D DFT.
- **shape** (*tuple*) – The output shape.

Returns **out**

Return type array

Notes

This function is a wrapper of numpy.fft.irfft2. The DFT is made on the two last axis.

class Operator
An abstract base class for linear operators.

abstract forward (*point*)
Return $H \cdot x$

abstract adjoint (*point*)
Return $H^* \cdot e$

fwback (*point*)
Return $H^* H \cdot x$

T (*point*)
Return $H^* \cdot e$

class Conv2 (*ir*, *shape*)
2D convolution on image.

Does not suppose periodic or circular condition.

imp_resp
The impulse response.

Type array

shape
The shape of the input image.

Type tuple of int

freq_resp
The frequency response of shape *shape*.

Type array

Notes

Use fft internally for fast computation. The `forward` methods is equivalent to convolve2d with "valid" boudary condition and `adjoint` is equivalent to convolve2d with "full" boundary condition with zero filling.

__init__(*ir*, *shape*)
2D convolution on image.

Parameters

- **ir** (*array*) – The impulse response.
- **shape** (*tuple of int*) – The shape of the input image.

forward(*point*)
Return $H \cdot x$

adjoint(*point*)
Return $H^T \cdot e$

fwback(*point*)
Return $H^T H \cdot x$

class Diff(*axis*)
Difference operator.

Compute the first-order differences along an axis.

axis
The axis along which the differences is performed.

Type int

Notes

Use *numpy.diff* and implement the correct adjoint, with *numpy.diff* also.

__init__(*axis*)
First-order differences operator.

Parameters **axis** (*int*) – the axis along which to perform the diff.

response(*ndim*)
Return the equivalent impulse response.

The result of *forward* method is equivalent with "valid" convolution with this impulse response. The adjoint operator corresponds the "full" convolution with the flipped impulse response.

freq_response (*ndim*, *shape*)
The frequency response.

forward (*point*)
Return $H \cdot x$

adjoint (*point*)
Return $H^? \cdot e$

ir2fr (*imp_resp*, *shape*, *center=None*, *real=True*)
Return the frequency response from impulse responses.

This function make the necessary correct zero-padding, zero convention, correct DFT etc. to compute the frequency response from impulse responses (IR).

The IR array is supposed to have the origin in the middle of the array.

The Fourier transform is performed on the last *len(shape)* dimensions.

Parameters

- **imp_resp** (*array*) – The impulse responses.
- **shape** (*tuple of int*) – A tuple of integer corresponding to the target shape of the frequency responses, without hermitian property. *len(shape)* >= *ndarray.ndim*. The DFT is performed on the *len(shape)* last axis of ndarray.
- **center** (*tuple of int, optional*) – The origin index of the impulse response. The middle by default.
- **real** (*boolean, optional*) – If True, imp_resp is supposed real, the hermitian property is used with rfftn DFT and the output has *shape[-1] / 2 + 1* elements on the last axis.

Returns **out** – The frequency responses of shape *shape* on the last *len(shape)* dimensions.

Return type array

Notes

- The output is returned as C-contiguous array.
- For convolution, the result have to be used with unitary discrete Fourier transform for the signal (*norm="ortho"* of fft).
- DFT are always performed on last axis for efficiency (C-order array).

5.8 How to extend Q-MM

To extend Q-MM you must subclass *qmm.BaseObjective* or *qmm.Loss*. Then *qmm.mmmg()* and *qmm.mmcg()* should work as expected.

Feel free to contribute with issues or pull request.

BIBLIOGRAPHY

[a] D. Geman and G. Reynolds, “Constrained restoration and the recovery of discontinuities,” IEEE Trans. Pattern Anal. Machine Intell., vol. 14, no. 3, pp. 367–383, Mar. 1992, doi: 10.1109/34.120331.

[b] D. Geman and C. Yang, “Nonlinear image recovery with half-quadratic regularization,” IEEE Trans. on Image Process., vol. 4, no. 7, pp. 932–946, Jul. 1995, doi: 10.1109/83.392335.

[c] M. Allain, J. Idier, and Y. Goussard, “On Global and Local Convergence of Half-Quadratic Algorithms,” IEEE Trans. on Image Processing, vol. 15, no. 5, p. 13, 2006.

[d] C. Labat and J. Idier, “Convergence of Conjugate Gradient Methods with a Closed-Form Stepsize Formula,” J Optim Theory Appl, p. 18, 2008.

[e] E. Chouzenoux, J. Idier, and S. Moussaoui, “A Majorize-Minimize Strategy for Subspace Optimization Applied to Image Restoration,” IEEE Trans. on Image Process., vol. 20, no. 6, pp. 1517–1528, Jun. 2011, doi: 10.1109/TIP.2010.2103083.

PYTHON MODULE INDEX

O

operators, 30

INDEX

Symbols

__init__() (*BaseObjective method*), 20
__init__() (*Conv2 method*), 32
__init__() (*Diff method*), 32
__init__() (*GemanMcClure method*), 30
__init__() (*HebertLeahy method*), 29
__init__() (*Huber method*), 29
__init__() (*Hyperbolic method*), 29
__init__() (*Loss method*), 28
__init__() (*MixedObjective method*), 27
__init__() (*Objective method*), 21
__init__() (*OptimizeResult method*), 19
__init__() (*QuadObjective method*), 22
__init__() (*Square method*), 28
__init__() (*TruncSquareApprox method*), 30
__init__() (*Vmax method*), 26
__init__() (*Vmin method*), 25

A

adjoint() (*Conv2 method*), 32
adjoint() (*Diff method*), 33
adjoint() (*Operator method*), 31
axis (*Diff attribute*), 32

B

BaseObjective (*class in qmm*), 19

C

calc_objv (*BaseObjective attribute*), 19
constant() (*QuadObjective property*), 23
Conv2 (*class in operators*), 31

D

dft2() (*in module operators*), 30
Diff (*class in operators*), 32

F

forward() (*Conv2 method*), 32
forward() (*Diff method*), 33
forward() (*Operator method*), 31
freq_resp (*Conv2 attribute*), 31
freq_response() (*Diff method*), 32
fwback() (*Conv2 method*), 32
fwback() (*Operator method*), 31

G

GemanMcClure (*class in qmm*), 30
gr_coeffs() (*Loss method*), 28
gr_coeffs() (*Objective method*), 22
gradient() (*BaseObjective method*), 20
gradient() (*GemanMcClure method*), 30
gradient() (*HebertLeahy method*), 30
gradient() (*Huber method*), 29
gradient() (*Hyperbolic method*), 29
gradient() (*Loss method*), 28
gradient() (*MixedObjective method*), 27
gradient() (*Objective method*), 21
gradient() (*QuadObjective method*), 23
gradient() (*Square method*), 28
gradient() (*TruncSquareApprox method*), 30
gradient() (*Vmax method*), 26
gradient() (*Vmin method*), 25
gy_coeffs() (*Loss method*), 28
gy_coeffs() (*Objective method*), 22

H

HebertLeahy (*class in qmm*), 29
Huber (*class in qmm*), 29
hyper (*BaseObjective attribute*), 19
Hyperbolic (*class in qmm*), 29

I

idft2() (*in module operators*), 31
imp_resp (*Conv2 attribute*), 31
insert() (*MixedObjective method*), 27

ir2fr() (*in module operators*), 33

L

lastgv (*BaseObjective attribute*), 20
lastv (*BaseObjective attribute*), 20
lastv() (*MixedObjective property*), 27
lcg() (*in module qmm*), 18
Loss (*class in qmm*), 27

M

MixedObjective (*class in qmm*), 27
mmcg() (*in module qmm*), 16
mmmg() (*in module qmm*), 17
module
    operators, 30

N

name (*BaseObjective attribute*), 19
norm_mat_major() (*BaseObjective method*), 20
norm_mat_major() (*Objective method*), 21
norm_mat_major() (*QuadObjective method*), 24
norm_mat_major() (*Vmax method*), 26
norm_mat_major() (*Vmin method*), 25

O

Objective (*class in qmm*), 20
Operator (*class in operators*), 31
operator() (*BaseObjective method*), 20
operator() (*Objective method*), 21
operator() (*QuadObjective method*), 23
operator() (*Vmax method*), 26
operator() (*Vmin method*), 25
operators
    module, 30
OptimizeResult (*class in qmm*), 18

Q

QuadObjective (*class in qmm*), 22

R

response() (*Diff method*), 32

S

shape (*Conv2 attribute*), 31
Square (*class in qmm*), 28

T

T() (*Operator method*), 31
TruncSquareApprox (*class in qmm*), 30

V

value() (*BaseObjective method*), 20
value() (*GemanMcClure method*), 30
value() (*HebertLeahy method*), 29
value() (*Huber method*), 29
value() (*Hyperbolic method*), 29
value() (*Loss method*), 28
value() (*MixedObjective method*), 27
value() (*Objective method*), 21
value() (*QuadObjective method*), 23
value() (*Square method*), 28
value() (*TruncSquareApprox method*), 30
value() (*Vmax method*), 26
value() (*Vmin method*), 25
value_hessp() (*QuadObjective method*), 24
value_residual() (*QuadObjective method*), 24
Vmax (*class in qmm*), 25
Vmin (*class in qmm*), 25
VtB_data() (*QuadObjective property*), 23