
Fides Documentation

Release 0.7.2

The Fides developers

Nov 26, 2021

ABOUT

1	About Fides	3
1.1	Features	3
2	Fides Python API	5
2.1	Fides	6
2.2	Minimization	6
2.3	Trust Region Step Evaluation	9
2.4	Subproblem Solvers	10
2.5	Hessian Update Strategies	13
2.6	Logging	20
2.7	Constants	20
3	Indices and tables	25
	Python Module Index	27
	Index	29

Version: 0.7.2

Source code: <https://github.com/Fides-dev/fides>

ABOUT FIDES

Fides implements an Interior Trust Region Reflective for boundary constrained optimization problems based on the papers [ColemanLi1994] and [ColemanLi1996]. Accordingly, Fides is named after the Roman goddess of trust and reliability. In contrast to other optimizers, Fides solves the full trust -region subproblem exactly, which can yields higher quality proposal steps, but is computationally more expensive. This makes Fides particularly attractive for optimization problems with objective functions that are computationally expensive to evaluate and the computational cost of solving the trust -region subproblem is negligible.

1.1 Features

- Boundary constrained interior trust-region optimization
- Recursive Reflective and Truncated constraint management
- Full and 2D subproblem solution solvers
- BFGS, DFP and SR1 Hessian Approximations

FIDES PYTHON API

Modules

<i>fides</i>	Fides Fides is an interior trust-region reflective optimizer
<i>fides.minimize</i>	Minimization This module performs the optimization given a step proposal.
<i>fides.trust_region</i>	Trust Region Step Evaluation This module provides the machinery to evaluate different trust-region(-reflective) step proposals and select among them based on to their performance according to the quadratic approximation of the objective function
<i>fides.subproblem</i>	Subproblem Solvers This module provides the machinery to solve 1- and N-dimensional trust-region subproblems.
<i>fides.hessian_approximation</i>	Hessian Update Strategies This module provides various generic Hessian approximation strategies that can be employed when the calculating the exact Hessian or an approximation is computationally too demanding.
<i>fides.logging</i>	Logging This module provides the machinery that is used to display progress of the optimizer as well as debugging information
<i>fides.constants</i>	Constants This module provides a central place to define native python enums and constants that are used in multiple other modules

2.1 Fides

Fides is an interior trust-region reflective optimizer

2.2 Minimization

This module performs the optimization given a step proposal.

Classes Summary

<i>FunEvaluator</i> (fun, nargout, resfun, funargs)	
<i>Funout</i> (fval, grad, x[, hess, res, sres])	
<i>Optimizer</i> (fun, ub, lb[, verbose, options, ...])	Performs optimization

Functions Summary

<i>validate_options</i> (options)	Check if the chosen options are valid
-----------------------------------	---------------------------------------

Classes

```
class fides.minimize.FunEvaluator(fun, nargout, resfun, funargs)
```

```
    __init__(fun, nargout, resfun, funargs)
```

```
class fides.minimize.Funout(fval, grad, x, hess=None, res=None, sres=None)
```

```
    __init__(fval, grad, x, hess=None, res=None, sres=None)
```

```
class fides.minimize.Optimizer(fun, ub, lb, verbose=20, options=None, funargs=None,
                               hessian_update=None, resfun=False)
```

Performs optimization

Variables

- **fevaler** – FunctionEvaluator instance
- **lb** – Lower optimization boundaries
- **ub** – Upper optimization boundaries
- **options** – Options that configure convergence checks
- **delta_iter** – Trust region radius that was used for the current step
- **delta** – Updated trust region radius
- **x** – Current optimization variables
- **fval** – Objective function value at x

- **grad** – Objective function gradient at x
- **x_min** – Optimal optimization variables
- **fval_min** – Objective function value at x_{\min}
- **grad_min** – Objective function gradient at x_{\min}
- **hess** – Objective function Hessian (approximation) at x
- **hessian_update** – Object that performs hessian updates
- **starttime** – Time at which optimization was started
- **iteration** – Current iteration
- **converged** – Flag indicating whether optimization has converged
- **exitflag** – ExitFlag to indicate reason for termination
- **verbose** – Verbosity level for logging
- **logger** – logger instance

__init__(*fun, ub, lb, verbose=20, options=None, funargs=None, hessian_update=None, resfun=False*)
Create an optimizer object

Parameters

- **fun** ([typing.Callable](#)) – This is the objective function, if no *hessian_update* is provided, this function must return a tuple (fval, grad), otherwise this function must return a tuple (fval, grad, Hessian). If the argument *resfun* is True, this function must return a tuple (res, sres) instead, where *sres* is the derivative of res.
- **ub** ([numpy.ndarray](#)) – Upper optimization boundaries. Individual entries can be set to `np.inf` for respective variable to have no upper bound
- **lb** ([numpy.ndarray](#)) – Lower optimization boundaries. Individual entries can be set to `-np.inf` for respective variable to have no lower bound
- **verbose** ([typing.Optional\[int\]](#)) – Verbosity level, pick from logging.[DEBUG,INFO,WARNING,ERROR]
- **options** ([typing.Optional\[typing.Dict\]](#)) – Options that control termination of optimization. See *minimize* for details.
- **funargs** ([typing.Optional\[typing.Dict\]](#)) – Additional keyword arguments that are to be passed to *fun* for evaluation
- **hessian_update** ([typing.Optional\[fides.hessian_approximation.HessianApproximation\]](#)) – Subclass of *fides.hessian_update.HessianApproximation* that performs the hessian updates in every iteration.
- **resfun** ([bool](#)) – Boolean flag indicating whether *fun* returns function values (False, default) or residuals (True).

check_continue()

Checks whether minimization should continue based on convergence, iteration count and remaining computational budget

Return type [bool](#)

Returns flag indicating whether minimization should continue

check_convergence(*step, funout*)

Check whether optimization has converged.

Parameters

- **step** (`fides.steps.Step`) – update to optimization variables
- **funout** (`fides.minimize.Funout`) – Function output generated by a `FunEvaluator`

Return type `None`**check_finite**(`funout=None`)

Checks whether objective function value, gradient and Hessian (approximation) have finite values and optimization can continue.

Parameters **funout** (`typing.Optional[fides.minimize.Funout]`) – Function output generated by a `FunEvaluator`

Raises `RuntimeError` if any of the variables have non-finite entries

check_in_bounds(`x=None`)

Checks whether the current optimization variables are all within the specified boundaries

Raises `RuntimeError` if any of the variables are not within boundaries

get_affine_scaling()

Computes the vector v and dv , the diagonal of its Jacobian. For the definition of v , see Definition 2 in [Coleman-Li1994]

Return type `typing.Tuple[numpy.ndarray, numpy.ndarray]`

Returns v scaling vector dv diagonal of the Jacobian of v wrt x

log_header()

Prints the header for diagnostic information, should complement `Optimizer.log_step()`.

log_step(`accepted, step, funout`)

Prints diagnostic information about the current step to the log

Parameters

- **accepted** (`bool`) – flag indicating whether the current step was accepted
- **step** (`fides.steps.Step`) – proposal step
- **funout** (`fides.minimize.Funout`) – Function output generated by a `FunEvaluator`

log_step_initial()

Prints diagnostic information about the initial step to the log

make_non_degenerate(`eps=2.220446049250313e-14`)

Ensures that x is non-degenerate, this should only be necessary for initial points.

Parameters **eps** – degeneracy threshold

Return type `None`**minimize**(`x0, start_id=None`)

Minimize the objective function using the interior trust-region reflective algorithm described by [ColemanLi1994] and [ColemanLi1996] Convergence with respect to function value is achieved when $|f_{k+1} - f_k| < \text{options}[fatol] - f_k \text{ options}[fritol]$. Similarly, convergence with respect to optimization variables is achieved when $\|x_{k+1} - x_k\| < \text{options}[xtol] x_k$ (note that this is checked in transformed coordinates that account for distance to boundaries). Convergence with respect to the gradient is achieved when $\|g_k\| < \text{options}[gatol]$ or $\|g_k\| < \text{options}[grtol] * f_k$. Other than that, optimization can be terminated when iterations exceed `options[maxiter]` or the elapsed time is expected to exceed `options[maxtime]` on the next iteration.

Parameters **x0** (`numpy.ndarray`) – initial guess

Returns fval: final function value, x: final optimization variable values, grad: final gradient, hess: final Hessian (approximation)

track_minimum(*funout*)

Function that tracks the optimization variables that have minimal function value independent of whether the step is accepted or not.

Parameters *funout* (*fides.minimize.Funout*) – Function output generated by a *FunEvaluator* evaluated at new x

Return type *None*

update(*step*, *funout_new*, *funout*)

Update self according to employed step

Parameters

- **step** (*fides.steps.Step*) – Employed step
- **funout** (*fides.minimize.Funout*) – Function output generated by a *FunEvaluator* for new variables before step is taken
- **funout_new** (*fides.minimize.Funout*) – Function output generated by a *FunEvaluator* for new variables after step is taken

Return type *None*

update_tr_radius(*funout*, *step*, *dv*)

Update the trust region radius

Parameters

- **funout** (*fides.minimize.Funout*) – Function output generated by a *FunEvaluator* for new variables after step is taken
- **step** (*fides.steps.Step*) – step
- **dv** (*numpy.ndarray*) – derivative of scaling vector v wrt x

Return type *bool*

Returns flag indicating whether the proposed step should be accepted

Functions

fides.minimize.validate_options(*options*)

Check if the chosen options are valid

2.3 Trust Region Step Evaluation

This module provides the machinery to evaluate different trust-region(-reflective) step proposals and select among them based on to their performance according to the quadratic approximation of the objective function

Functions Summary

<code>trust_region(x, g, hess, scaling, delta, dv, ...)</code>	Compute a step according to the solution of the trust-region subproblem.
--	--

Functions

`fides.trust_region.trust_region(x, g, hess, scaling, delta, dv, theta, lb, ub, subspace_dim, stepback_strategy, logger)`

Compute a step according to the solution of the trust-region subproblem. If step-back is necessary, gradient and reflected trust region step are also evaluated in terms of their performance according to the local quadratic approximation

Parameters

- **x** (`numpy.ndarray`) – Current values of the optimization variables
- **g** (`numpy.ndarray`) – Objective function gradient at x
- **hess** (`numpy.ndarray`) – (Approximate) objective function Hessian at x
- **scaling** (`scipy.sparse.csc.csc_matrix`) – Scaling transformation according to distance to boundary
- **delta** (`float`) – Trust region radius, note that this applies after scaling transformation
- **dv** (`numpy.ndarray`) – derivative of scaling transformation
- **theta** (`float`) – parameter regulating stepback
- **lb** (`numpy.ndarray`) – lower optimization variable boundaries
- **ub** (`numpy.ndarray`) – upper optimization variable boundaries
- **subspace_dim** (`fides.constants.SubSpaceDim`) – Subspace dimension in which the subproblem will be solved. Larger subspaces require more compute time but can yield higher quality step proposals.
- **stepback_strategy** (`fides.constants.StepBackStrategy`) – Strategy that is applied when the proposed step exceeds the optimization boundary.
- **logger** (`logging.Logger`) – logging.Logger instance to be used for logging

Return type `fides.steps.Step`

Returns s: proposed step,

2.4 Subproblem Solvers

This module provides the machinery to solve 1- and N-dimensional trust-region subproblems.

Functions Summary

<code>dsecular(lam, w, eigvals, eigvecs, delta)</code>	Derivative of the secular equation
<code>dslam(lam, w, eigvals, eigvecs)</code>	Computes the derivative of the solution $s(\lambda)$ with respect to lambda, where s is the subproblem solution according to
<code>get_1d_trust_region_boundary_solution(B, g, ...)</code>	
<code>quadratic_form(Q, p, x)</code>	Computes the quadratic form $x^T Q x + x^T p$
<code>secular(lam, w, eigvals, eigvecs, delta)</code>	Secular equation
<code>slam(lam, w, eigvals, eigvecs)</code>	Computes the solution $s(\lambda)$ as subproblem solution according to
<code>solve_1d_trust_region_subproblem(B, g, s, ...)</code>	Solves the special case of a one-dimensional subproblem
<code>solve_nd_trust_region_subproblem(B, g, delta)</code>	This function exactly solves the n-dimensional subproblem.

Functions

`fides.subproblem.dsecular(lam, w, eigvals, eigvecs, delta)`

Derivative of the secular equation

$$\phi(\lambda) = \frac{1}{\|s\|} - \frac{1}{\Delta}$$

with respect to λ

Parameters

- **lam** (`float`) – λ
- **w** (`numpy.ndarray`) – precomputed eigenvector coefficients for -g
- **eigvals** (`numpy.ndarray`) – precomputed eigenvalues of B
- **eigvecs** (`numpy.ndarray`) – precomputed eigenvectors of B
- **delta** (`float`) – trust region radius Δ

Returns $\frac{\partial \phi(\lambda)}{\partial \lambda}$

`fides.subproblem.dslam(lam, w, eigvals, eigvecs)`

Computes the derivative of the solution $s(\lambda)$ with respect to lambda, where s is the subproblem solution according to

$$-(B + \lambda I)s = g$$

Parameters

- **lam** (`float`) – λ
- **w** (`numpy.ndarray`) – precomputed eigenvector coefficients for -g
- **eigvals** (`numpy.ndarray`) – precomputed eigenvalues of B
- **eigvecs** (`numpy.ndarray`) – precomputed eigenvectors of B

Returns $\frac{\partial s(\lambda)}{\partial \lambda}$

`fides.subproblem.get_1d_trust_region_boundary_solution(B, g, s, s0, delta)`

`fides.subproblem.quadratic_form(Q, p, x)`

Computes the quadratic form $x^T Q x + x^T p$

Parameters

- **Q** (`numpy.ndarray`) – Matrix
- **p** (`numpy.ndarray`) – Vector
- **x** (`numpy.ndarray`) – Input

Return type `float`**Returns** Value of form`fides.subproblem.secular(lam, w, eigvals, eigvecs, delta)`

Secular equation

$$\phi(\lambda) = \frac{1}{\|s\|} - \frac{1}{\Delta}$$

Subproblem solutions are given by the roots of this equation

Parameters

- **lam** (`float`) – λ
- **w** (`numpy.ndarray`) – precomputed eigenvector coefficients for -g
- **eigvals** (`numpy.ndarray`) – precomputed eigenvalues of B
- **eigvecs** (`numpy.ndarray`) – precomputed eigenvectors of B
- **delta** (`float`) – trust region radius Δ

Returns $\phi(\lambda)$ `fides.subproblem.slam(lam, w, eigvals, eigvecs)`Computes the solution $s(\lambda)$ as subproblem solution according to

$$-(B + \lambda I)s = g$$

Parameters

- **lam** (`float`) – λ
- **w** (`numpy.ndarray`) – precomputed eigenvector coefficients for -g
- **eigvals** (`numpy.ndarray`) – precomputed eigenvalues of B
- **eigvecs** (`numpy.ndarray`) – precomputed eigenvectors of B

Return type `numpy.ndarray`**Returns** $s(\lambda)$ `fides.subproblem.solve_1d_trust_region_subproblem(B, g, s, delta, s0)`

Solves the special case of a one-dimensional subproblem

Parameters

- **B** (`numpy.ndarray`) – Hessian of the quadratic subproblem
- **g** (`numpy.ndarray`) – Gradient of the quadratic subproblem
- **s** (`numpy.ndarray`) – Vector defining the one-dimensional search direction
- **delta** (`float`) – Norm boundary for the solution of the quadratic subproblem
- **s0** (`numpy.ndarray`) – reference point from where search is started, also counts towards norm of step

Return type `numpy.ndarray`

Returns Proposed step-length

`fides.subproblem.solve_nd_trust_region_subproblem(B, g, delta, logger=None)`

This function exactly solves the n-dimensional subproblem.

$$\operatorname{argmin}_s \{s^T B s + s^T g = 0 : \|s\| \leq \Delta, s \in \mathbb{R}^n\}$$

The solution is characterized by the equation $-(B + \lambda I)s = g$. If B is positive definite, the solution can be obtained by $\lambda = 0$ if $Bs = -g$ satisfies $\|s\| \leq \Delta$. If B is indefinite or $Bs = -g$ satisfies $\|s\| > \Delta$ and an appropriate λ has to be identified via 1D rootfinding of the secular equation

$$\phi(\lambda) = \frac{1}{\|s(\lambda)\|} - \frac{1}{\Delta} = 0$$

with $s(\lambda)$ computed according to an eigenvalue decomposition of B. The eigenvalue decomposition, although being more expensive than a cholesky decomposition, has the advantage that eigenvectors are invariant to changes in λ and eigenvalues are linear in λ , so factorization only has to be performed once. We perform the linesearch via Newton's algorithm and Brent-Q as fallback. The hard case is treated separately and serves as general fallback.

Parameters

- **B** (`numpy.ndarray`) – Hessian of the quadratic subproblem
- **g** (`numpy.ndarray`) – Gradient of the quadratic subproblem
- **delta** (`float`) – Norm boundary for the solution of the quadratic subproblem
- **logger** (`typing.Optional[logging.Logger]`) – Logger instance to be used for logging

Return type `typing.Tuple[numpy.ndarray, str]`

Returns `s`: Selected step, `step_type`: Type of solution that was obtained

2.5 Hessian Update Strategies

This module provides various generic Hessian approximation strategies that can be employed when the calculating the exact Hessian or an approximation is computationally too demanding.

Classes Summary

<code>BB([init_with_hess])</code>	Broydens "bad" method as introduced in [Broyden 1965](https://doi.org/10.1090%2FS0025-5718-1965-0198670-6).
<code>BFGS([init_with_hess, enforce_curv_cond])</code>	Broyden-Fletcher-Goldfarb-Shanno update strategy.
<code>BG([init_with_hess])</code>	Broydens "good" method as introduced in [Broyden 1965](https://doi.org/10.1090%2FS0025-5718-1965-0198670-6).
<code>Broyden(phi[, init_with_hess, enforce_curv_cond])</code>	BroydenClass Update scheme as described in [Nocedal & Wright](http://dx.doi.org/10.1007/b98874) Chapter 6.3.
<code>DFP([init_with_hess, enforce_curv_cond])</code>	Davidon-Fletcher-Powell update strategy.
<code>FX([happ, hybrid_tol])</code>	
<code>GNSBFGS([hybrid_tol, enforce_curv_cond])</code>	

continues on next page

Table 6 – continued from previous page

<i>HessianApproximation</i> ([init_with_hess])	Abstract class from which Hessian update strategies should subclass
<i>HybridApproximation</i> ([happ])	
<i>HybridFixed</i> ([happ, switch_iteration])	
<i>HybridFraction</i> ([happ, switch_threshold])	
<i>HybridSwitchApproximation</i> ([happ])	
<i>IterativeHessianApproximation</i> ([init_with_hess])	Iterative update schemes that only use s and y values for update.
<i>SRI</i> ([init_with_hess])	Symmetric Rank 1 update strategy as described in [Nocedal & Wright](http://dx.doi.org/10.1007/b98874) Chapter 6.2.
<i>SSM</i> ([phi, enforce_curv_cond])	Structured Secant Method as introduced by [Dennis et al 1989](https://doi.org/10.1007/BF00962795), which is compatible with BFGS, DFP update schemes.
<i>StructuredApproximation</i> ([phi, force_curv_cond])	en-
<i>TSSM</i> ([phi, enforce_curv_cond])	Totally Structured Secant Method as introduced by [Huschens 1994](https://doi.org/10.1137/0804005), which uses a self-adjusting update method for the second order term.

Functions Summary

<i>broyden_class_update</i> (y, s, mat[, phi, ...])	Scale free implementation of the broyden class update scheme.
---	---

Classes

class fides.hessian_approximation.**BB**(init_with_hess=False)

Broydens “bad” method as introduced in [Broyden 1965](<https://doi.org/10.1090%2FS0025-5718-1965-0198670-6>). This is a rank 1 update strategy that does not preserve symmetry or positive definiteness.

This scheme only works with a function that returns (fval, grad)

update(s, y)

Update the Hessian approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization variables
- **y** (`numpy.ndarray`) – step in gradient

Return type `None`

class fides.hessian_approximation.**BFGS**(init_with_hess=False, enforce_curv_cond=True)

Broyden-Fletcher-Goldfarb-Shanno update strategy. This is a rank 2 update strategy that preserves symmetry and positive-semidefiniteness.

This scheme only works with a function that returns (fval, grad)

```
__init__(init_with_hess=False, enforce_curv_cond=True)
```

Create a Hessian update strategy instance

Parameters **init_with_hess** (`typing.Optional[bool]`) – Whether the hybrid update strategy should be initialized according to the user-provided objective function

```
class fides.hessian_approximation.BG(init_with_hess=False)
```

Broydens “good” method as introduced in [Broyden 1965](<https://doi.org/10.1090%2FS0025-5718-1965-0198670-6>). This is a rank 1 update strategy that does not preserve symmetry or positive definiteness.

This scheme only works with a function that returns (fval, grad)

```
class fides.hessian_approximation.Broyden(phi, init_with_hess=False, enforce_curv_cond=True)
```

BroydenClass Update scheme as described in [Nocedal & Wright](<http://dx.doi.org/10.1007/b98874>) Chapter 6.3. This is a generalization of BFGS/DFP methods where the parameter *phi* controls the convex combination between the two. This is a rank 2 update strategy that preserves positive-semidefiniteness and symmetry (if $\phi \in [0, 1]$).

This scheme only works with a function that returns (fval, grad)

Parameters

- **phi** (`float`) – convex combination parameter interpolating between BFGS ($\phi==0$) and DFP ($\phi==1$).
- **enforce_curv_cond** (`typing.Optional[bool]`) – boolean that controls whether the employed broyden class update should attempt to preserve positive definiteness. If set to True, updates from steps that violate the curvature condition will be discarded.

```
__init__(phi, init_with_hess=False, enforce_curv_cond=True)
```

Create a Hessian update strategy instance

Parameters **init_with_hess** (`typing.Optional[bool]`) – Whether the hybrid update strategy should be initialized according to the user-provided objective function

```
class fides.hessian_approximation.DFP(init_with_hess=False, enforce_curv_cond=True)
```

Davidon-Fletcher-Powell update strategy. This is a rank 2 update strategy that preserves symmetry and positive-semidefiniteness.

This scheme only works with a function that returns (fval, grad)

```
__init__(init_with_hess=False, enforce_curv_cond=True)
```

Create a Hessian update strategy instance

Parameters **init_with_hess** (`typing.Optional[bool]`) – Whether the hybrid update strategy should be initialized according to the user-provided objective function

```
class fides.hessian_approximation.FX(happ=<fides.hessian_approximation.BFGS object>,
                                     hybrid_tol=0.2)
```

```
__init__(happ=<fides.hessian_approximation.BFGS object>, hybrid_tol=0.2)
```

Hybrid method HY2 as introduced by [Fletcher & Xu 1986](<https://doi.org/10.1093/imanum/7.3.371>). This approximation scheme employs a dynamic approximation as long as function values satisfy $\frac{f_k - f_{k+1}}{f_k} < \epsilon$ and employs the iterative scheme applied to the last dynamic approximation if not.

This scheme only works with a function that returns (fval, grad, hess)

Parameters **hybrid_tol** (`typing.Optional[float]`) – switch tolerance ϵ

get_mat()

Getter for the Hessian approximation

Return type `numpy.ndarray`

Returns Hessian approximation

update(*delta*, *gamma*, *r*, *rprev*, *hess*)

Update the Hessian approximation

Parameters

- **delta** (`numpy.ndarray`) – step in optimization variables
- **gamma** (`numpy.ndarray`) – step in gradient
- **r** (`numpy.ndarray`) – residuals after current step
- **rprev** (`numpy.ndarray`) – residuals before current step
- **hess** (`numpy.ndarray`) – user-provided (Gauss-Newton) Hessian approximation

Return type `None`

class `fides.hessian_approximation.GNSBFGS(hybrid_tol=1e-06, enforce_curv_cond=True)`

__init__(*hybrid_tol=1e-06, enforce_curv_cond=True*)

Hybrid Gauss-Newton Structured BFGS method as introduced by [Zhou & Chen 2010](<https://doi.org/10.1137/090748470>), which combines ideas of hybrid switching methods and structured secant methods.

This scheme only works with a function that returns (res, sres)

Parameters **hybrid_tol** (`float`) – switching tolerance that controls switching between update methods

update(*s*, *y*, *r*, *hess*, *yb*)

Update the structured approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization parameters
- **y** (`numpy.ndarray`) – step in gradient parameters
- **r** (`numpy.ndarray`) – residual vector
- **hess** (`numpy.ndarray`) – user-provided (Gauss-Newton) Hessian approximation
- **yb** (`numpy.ndarray`) – approximation to A^*s , where A is structured approximation matrix

Return type `None`

class `fides.hessian_approximation.HessianApproximation(init_with_hess=False)`

Abstract class from which Hessian update strategies should subclass

__init__(*init_with_hess=False*)

Create a Hessian update strategy instance

Parameters **init_with_hess** (`typing.Optional[bool]`) – Whether the hybrid update strategy should be initialized according to the user-provided objective function

get_diff()

Getter for the Hessian approximation update

Return type `numpy.ndarray`

Returns Hessian approximation update

get_mat()

Getter for the Hessian approximation

Return type `numpy.ndarray`

Returns Hessian approximation

init_mat(dim, hess=None)

Initializes this approximation instance and checks the dimensionality

Parameters

- **dim** (`int`) – dimension of optimization variables
- **hess** (`typing.Optional[numpy.ndarray]`) – user provided initialization

Return type `None`

set_mat(mat)

Setter for the Hessian approximation

Parameters **mat** (`numpy.ndarray`) – Hessian approximation

Return type `None`

class `fides.hessian_approximation.HybridApproximation(happ=<fides.hessian_approximation.BFGS object>)`

__init__(*happ=<fides.hessian_approximation.BFGS object>*)

Create a Hybrid Hessian update strategy that switches between an iterative approximation and a dynamic approximation

Parameters **happ** (`fides.hessian_approximation.IterativeHessianApproximation`) – Iterative Hessian Approximation

init_mat(dim, hess=None)

Initializes this approximation instance and checks the dimensionality

Parameters

- **dim** (`int`) – dimension of optimization variables
- **hess** (`typing.Optional[numpy.ndarray]`) – user provided initialization

class `fides.hessian_approximation.HybridFixed(happ=<fides.hessian_approximation.BFGS object>, switch_iteration=20)`

__init__(*happ=<fides.hessian_approximation.BFGS object>, switch_iteration=20*)

Switch from a dynamic approximation to the user provided iterative scheme after a fixed number of successive iterations without trust-region update. The switching is non-reversible. The iterative scheme is initialized and updated from the beginning, but only employed after the specified number of iterations.

This scheme only works with a function that returns (fval, grad, hess)

Parameters **switch_iteration** (`typing.Optional[int]`) – Number of iterations without trust region update after which switch occurs.

class `fides.hessian_approximation.HybridFraction(happ=<fides.hessian_approximation.BFGS object>, switch_threshold=0.8)`

__init__(*happ=<fides.hessian_approximation.BFGS object>, switch_threshold=0.8*)

Switch from a dynamic approximation to the user provided iterative scheme as soon as the fraction of iterations where the step is accepted but the trust region is not update exceeds the user provided threshold. Threshold check is only performed after 25 iterations. The switching is non-reversible. The iterative scheme is initialized and updated from the beginning, but only employed after the switching.

This scheme only works with a function that returns (fval, grad, hess)

Parameters **switch_threshold** (`typing.Optional[float]`) – Threshold for fraction of iterations where step is accepted but trust region is not updated, which when exceeded triggers switch of approximation.

class `fides.hessian_approximation.HybridSwitchApproximation`(*happ=<fides.hessian_approximation.BFGS object>*)

class `fides.hessian_approximation.IterativeHessianApproximation`(*init_with_hess=False*)

Iterative update schemes that only use s and y values for update.

update(*s, y*)

Update the Hessian approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization variables
- **y** (`numpy.ndarray`) – step in gradient

Return type `None`

class `fides.hessian_approximation.SR1`(*init_with_hess=False*)

Symmetric Rank 1 update strategy as described in [Nocedal & Wright](<http://dx.doi.org/10.1007/b98874>) Chapter 6.2. This is a rank 1 update strategy that preserves symmetry but does not preserve positive-semidefiniteness.

This scheme only works with a function that returns (fval, grad)

class `fides.hessian_approximation.SSM`(*phi=0.0, enforce_curv_cond=True*)

Structured Secant Method as introduced by [Dennis et al 1989](<https://doi.org/10.1007/BF00962795>), which is compatible with BFGS, DFP update schemes.

This scheme only works with a function that returns (res, sres)

update(*s, y, r, hess, yb*)

Update the structured approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization parameters
- **y** (`numpy.ndarray`) – step in gradient parameters
- **r** (`numpy.ndarray`) – residual vector
- **hess** (`numpy.ndarray`) – user-provided (Gauss-Newton) Hessian approximation
- **yb** (`numpy.ndarray`) – approximation to A^*s , where A is structured approximation matrix

Return type `None`

class `fides.hessian_approximation.StructuredApproximation`(*phi=0.0, enforce_curv_cond=True*)

__init__(*phi=0.0, enforce_curv_cond=True*)

This is the base class for structured secant methods (SSM). SSMs approximate the hessian by combining the Gauss-Newton component $C(x)$ and an iteratively updated component that approximates the difference S to the true Hessian.

Parameters

- **phi** (`typing.Optional[float]`) – convex combination parameter interpolating between BFGS (`phi==0`) and DFP (`phi==1`) update schemes.
- **enforce_curv_cond** (`typing.Optional[bool]`) – boolean that controls whether the employed broyden class update should attempt to preserve positive definiteness. If set to `True`, updates from steps that violate the curvature condition will be discarded.

init_mat(*dim*, *hess=None*)

Initializes this approximation instance and checks the dimensionality

Parameters

- **dim** (`int`) – dimension of optimization variables
- **hess** (`typing.Optional[numpy.ndarray]`) – user provided initialization

update(*s*, *y*, *r*, *hess*, *yb*)

Update the structured approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization parameters
- **y** (`numpy.ndarray`) – step in gradient parameters
- **r** (`numpy.ndarray`) – residual vector
- **hess** (`numpy.ndarray`) – user-provided (Gauss-Newton) Hessian approximation
- **yb** (`numpy.ndarray`) – approximation to $A*s$, where A is structured approximation matrix

Return type `None`

class `fides.hessian_approximation.TSSM(phi=0.0, enforce_curv_cond=True)`

Totally Structured Secant Method as introduced by [Huschens 1994](<https://doi.org/10.1137/0804005>), which uses a self-adjusting update method for the second order term.

This scheme only works with a function that returns (`res`, `sres`)

update(*s*, *y*, *r*, *hess*, *yb*)

Update the structured approximation

Parameters

- **s** (`numpy.ndarray`) – step in optimization parameters
- **y** (`numpy.ndarray`) – step in gradient parameters
- **r** (`numpy.ndarray`) – residual vector
- **hess** (`numpy.ndarray`) – user-provided (Gauss-Newton) Hessian approximation
- **yb** (`numpy.ndarray`) – approximation to $A*s$, where A is structured approximation matrix

Return type `None`

Functions

`fides.hessian_approximation.broyden_class_update(y, s, mat, phi=0.0, enforce_curv_cond=True)`
Scale free implementation of the broyden class update scheme.

Parameters

- **y** (`numpy.ndarray`) – difference in gradient
- **s** (`numpy.ndarray`) – search direction in previous step
- **mat** (`numpy.ndarray`) – current hessian approximation
- **phi** (`float`) – convex combination parameter, interpolates between BFGS (`phi=0`) and DFP (`phi=1`).
- **enforce_curv_cond** (`bool`) – boolean that controls whether the employed broyden class update should attempt to preserve positive definiteness. If set to `True`, updates from steps that violate the curvature condition will be discarded.

Return type `numpy.ndarray`

2.6 Logging

This module provides the machinery that is used to display progress of the optimizer as well as debugging information

var logger `logging.Logger` instance that can be used throughout fides

Functions Summary

<code>create_logger(level)</code>	Creates a logger instance.
-----------------------------------	----------------------------

Functions

`fides.logging.create_logger(level)`
Creates a logger instance. To avoid unnecessary locks during multithreading, different logger instance should be created for every

Parameters **level** (`int`) – logging level

Return type `logging.Logger`

Returns logger instance

2.7 Constants

This module provides a central place to define native python enums and constants that are used in multiple other modules

Classes Summary

<i>ExitFlag</i> (value)	Defines possible exitflag values for the optimizer to indicate why optimization exited.
<i>Options</i> (value)	Defines all the fields that can be specified in Options to Optimizer
<i>StepBackStrategy</i> (value)	Defines the possible choices of search refinement if proposed step reaches optimization boundary
<i>SubSpaceDim</i> (value)	Defines the possible choices of subspace dimension in which the subproblem will be solved.

Classes

class fides.constants.**ExitFlag**(value)

Defines possible exitflag values for the optimizer to indicate why optimization exited. Negative value indicate errors while positive values indicate convergence.

DELTA_TOO_SMALL = -5

Trust Region Radius too small to proceed

DID_NOT_RUN = 0

Optimizer did not run

EXCEEDED_BOUNDARY = -4

Exceeded specified boundaries

FTOL = 1

Converged according to fval difference

GTOL = 3

Converged according to gradient norm

MAXITER = -1

Reached maximum number of allowed iterations

MAXTIME = -2

Expected to reach maximum allowed time in next iteration

NOT_FINITE = -3

Encountered non-finite fval/grad/hess

XTOL = 2

Converged according to x difference

class fides.constants.**Options**(value)

Defines all the fields that can be specified in Options to **Optimizer**

DELTA_INIT = 'delta_init'

initial trust region radius

ETA = 'eta'

trust region increase threshold for trust region ratio

FATOL = 'fatol'

absolute tolerance for convergence based on fval

FRTOL = 'frtol'

relative tolerance for convergence based on fval

GAMMA1 = 'gamma1'
factor by which trust region radius will be decreased

GAMMA2 = 'gamma2'
factor by which trust region radius will be increased

GATOL = 'gatol'
absolute tolerance for convergence based on grad

GRTOL = 'grtol'
relative tolerance for convergence based on grad

HISTORY_FILE = 'history_file'
when set, statistics for each start will

MAXITER = 'maxiter'
maximum number of allowed iterations

MAXTIME = 'maxtime'
maximum amount of walltime in seconds

MU = 'mu'
acceptance threshold for trust region ratio

STEPBACK_STRAT = 'stepback_strategy'
method to use for stepback

SUBSPACE_DIM = 'subspace_solver'
trust region subproblem subspace

THETA_MAX = 'theta_max'
maximal fraction of step that would hit bounds

XTOL = 'xtol'
tolerance for convergence based on x

class fides.constants.**StepBackStrategy**(*value*)
Defines the possible choices of search refinement if proposed step reaches optimization boundary

MIXED = 'mixed'
mix reflections and truncations

REFINE = 'refine'
perform optimization to refine step

REFLECT = 'reflect'
recursive reflections at boundary

SINGLE_REFLECT = 'reflect_single'
single reflection at boundary

TRUNCATE = 'truncate'
truncate step at boundary and re-solve

class fides.constants.**SubSpaceDim**(*value*)
Defines the possible choices of subspace dimension in which the subproblem will be solved.

FULL = 'full'
Full \mathbb{R}^n

STEIHAUG = 'scg'
CG subspace via Steihaug's method

TWO = '2D'

Two dimensional Newton/Gradient subspace

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

f

- `fides`, [5](#)
- `fides.constants`, [20](#)
- `fides.hessian_approximation`, [13](#)
- `fides.logging`, [20](#)
- `fides.minimize`, [6](#)
- `fides.subproblem`, [10](#)
- `fides.trust_region`, [9](#)

Symbols

`__init__()` (*fides.hessian_approximation.BFGS* method), 15
`__init__()` (*fides.hessian_approximation.Broyden* method), 15
`__init__()` (*fides.hessian_approximation.DFP* method), 15
`__init__()` (*fides.hessian_approximation.FX* method), 15
`__init__()` (*fides.hessian_approximation.GNSBFGS* method), 16
`__init__()` (*fides.hessian_approximation.HessianApproximation* method), 16
`__init__()` (*fides.hessian_approximation.HybridApproximation* method), 17
`__init__()` (*fides.hessian_approximation.HybridFixed* method), 17
`__init__()` (*fides.hessian_approximation.HybridFraction* method), 17
`__init__()` (*fides.hessian_approximation.StructuredApproximation* method), 18
`__init__()` (*fides.minimize.FunEvaluator* method), 6
`__init__()` (*fides.minimize.Funout* method), 6
`__init__()` (*fides.minimize.Optimizer* method), 7

B

`BB` (class in *fides.hessian_approximation*), 14
`BFGS` (class in *fides.hessian_approximation*), 14
`BG` (class in *fides.hessian_approximation*), 15
`Broyden` (class in *fides.hessian_approximation*), 15
`broyden_class_update()` (in module *fides.hessian_approximation*), 20

C

`check_continue()` (*fides.minimize.Optimizer* method), 7
`check_convergence()` (*fides.minimize.Optimizer* method), 7
`check_finite()` (*fides.minimize.Optimizer* method), 8
`check_in_bounds()` (*fides.minimize.Optimizer* method), 8
`create_logger()` (in module *fides.logging*), 20

D

`DELTA_INIT` (*fides.constants.Options* attribute), 21
`DELTA_TOO_SMALL` (*fides.constants.ExitFlag* attribute), 21
`DFP` (class in *fides.hessian_approximation*), 15
`DID_NOT_RUN` (*fides.constants.ExitFlag* attribute), 21
`dsecular()` (in module *fides.subproblem*), 11
`dslam()` (in module *fides.subproblem*), 11

E

`ETA` (*fides.constants.Options* attribute), 21
`EXCEEDED_BOUNDARY` (*fides.constants.ExitFlag* attribute), 21
`ExitFlag` (class in *fides.constants*), 21

F

`FATOL` (*fides.constants.Options* attribute), 21
`fides`
 module, 5
 fides.constants
 module, 20
 fides.hessian_approximation
 module, 13
 fides.logging
 module, 20
 fides.minimize
 module, 6
 fides.subproblem
 module, 10
 fides.trust_region
 module, 9
`FRTOL` (*fides.constants.Options* attribute), 21
`FTOL` (*fides.constants.ExitFlag* attribute), 21
`FULL` (*fides.constants.SubSpaceDim* attribute), 22
`FunEvaluator` (class in *fides.minimize*), 6
`Funout` (class in *fides.minimize*), 6
`FX` (class in *fides.hessian_approximation*), 15

G

`GAMMA1` (*fides.constants.Options* attribute), 21
`GAMMA2` (*fides.constants.Options* attribute), 22

GATOL (*fides.constants.Options* attribute), 22
 get_1d_trust_region_boundary_solution() (in module *fides.subproblem*), 11
 get_affine_scaling() (*fides.minimize.Optimizer* method), 8
 get_diff() (*fides.hessian_approximation.HessianApproximation* method), 16
 get_mat() (*fides.hessian_approximation.FX* method), 15
 get_mat() (*fides.hessian_approximation.HessianApproximation* method), 16
 GNSBFGS (class in *fides.hessian_approximation*), 16
 GRTOL (*fides.constants.Options* attribute), 22
 GTOL (*fides.constants.ExitFlag* attribute), 21

H

HessianApproximation (class in *fides.hessian_approximation*), 16
 HISTORY_FILE (*fides.constants.Options* attribute), 22
 HybridApproximation (class in *fides.hessian_approximation*), 17
 HybridFixed (class in *fides.hessian_approximation*), 17
 HybridFraction (class in *fides.hessian_approximation*), 17
 HybridSwitchApproximation (class in *fides.hessian_approximation*), 18

I

init_mat() (*fides.hessian_approximation.HessianApproximation* method), 17
 init_mat() (*fides.hessian_approximation.HybridApproximation* method), 17
 init_mat() (*fides.hessian_approximation.StructuredApproximation* method), 19
 IterativeHessianApproximation (class in *fides.hessian_approximation*), 18

L

log_header() (*fides.minimize.Optimizer* method), 8
 log_step() (*fides.minimize.Optimizer* method), 8
 log_step_initial() (*fides.minimize.Optimizer* method), 8

M

make_non_degenerate() (*fides.minimize.Optimizer* method), 8
 MAXITER (*fides.constants.ExitFlag* attribute), 21
 MAXITER (*fides.constants.Options* attribute), 22
 MAXTIME (*fides.constants.ExitFlag* attribute), 21
 MAXTIME (*fides.constants.Options* attribute), 22
 minimize() (*fides.minimize.Optimizer* method), 8
 MIXED (*fides.constants.StepBackStrategy* attribute), 22
 module

fides, 5
fides.constants, 20
fides.hessian_approximation, 13
fides.logging, 20
fides.minimize, 6
fides.subproblem, 10
fides.trust_region, 9

MU (*fides.constants.Options* attribute), 22

N

NOT_FINITE (*fides.constants.ExitFlag* attribute), 21

O

Optimizer (class in *fides.minimize*), 6
 Options (class in *fides.constants*), 21

Q

quadratic_form() (in module *fides.subproblem*), 11

R

REFINE (*fides.constants.StepBackStrategy* attribute), 22
 REFLECT (*fides.constants.StepBackStrategy* attribute), 22

S

secular() (in module *fides.subproblem*), 12
 set_mat() (*fides.hessian_approximation.HessianApproximation* method), 17
 SINGLE_REFLECT (*fides.constants.StepBackStrategy* attribute), 22
 slam() (in module *fides.subproblem*), 12
 solve_1d_trust_region_subproblem() (in module *fides.subproblem*), 12
 solve_nd_trust_region_subproblem() (in module *fides.subproblem*), 13
 SR1 (class in *fides.hessian_approximation*), 18
 SSM (class in *fides.hessian_approximation*), 18
 STEIHAUG (*fides.constants.SubSpaceDim* attribute), 22
 STEPBACK_STRAT (*fides.constants.Options* attribute), 22
 StepBackStrategy (class in *fides.constants*), 22
 StructuredApproximation (class in *fides.hessian_approximation*), 18
 SUBSPACE_DIM (*fides.constants.Options* attribute), 22
 SubSpaceDim (class in *fides.constants*), 22

T

THETA_MAX (*fides.constants.Options* attribute), 22
 track_minimum() (*fides.minimize.Optimizer* method), 9
 TRUNCATE (*fides.constants.StepBackStrategy* attribute), 22
 trust_region() (in module *fides.trust_region*), 10
 TSSM (class in *fides.hessian_approximation*), 19
 TWO (*fides.constants.SubSpaceDim* attribute), 22

U

`update()` (*fides.hessian_approximation.BB method*), 14
`update()` (*fides.hessian_approximation.FX method*), 16
`update()` (*fides.hessian_approximation.GNSBFGS method*), 16
`update()` (*fides.hessian_approximation.IterativeHessianApproximation method*), 18
`update()` (*fides.hessian_approximation.SSM method*), 18
`update()` (*fides.hessian_approximation.StructuredApproximation method*), 19
`update()` (*fides.hessian_approximation.TSSM method*), 19
`update()` (*fides.minimize.Optimizer method*), 9
`update_tr_radius()` (*fides.minimize.Optimizer method*), 9

V

`validate_options()` (*in module fides.minimize*), 9

X

`XTOL` (*fides.constants.ExitFlag attribute*), 21
`XTOL` (*fides.constants.Options attribute*), 22