
Fides Documentation

Release 0.2.1

The Fides developers

Nov 20, 2020

ABOUT

1	About Fides	3
1.1	Features	3
2	Fides Python API	5
2.1	Fides	5
2.2	Minimization	5
2.3	Trust Region Steps	9
2.4	Subproblem Solvers	15
2.5	Hessian Update Strategies	18
2.6	Logging	18
2.7	Constants	19
3	Indices and tables	21
	Python Module Index	23
	Index	25

Version: 0.2.1

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 yield 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
<i>fides.minimize</i>	Minimization
<i>fides.trust_region</i>	Trust Region Steps
<i>fides.subproblem</i>	Subproblem Solvers
<i>fides.hessian_approximation</i>	Hessian Update Strategies
<i>fides.logging</i>	Logging
<i>fides.constants</i>	Constants

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>Optimizer</i> (fun, ub, lb[, verbose, options, ...])	Performs optimization
---	-----------------------

Classes

class `fides.minimize.Optimizer` (*fun, ub, lb, verbose=10, options=None, funargs=None, hessian_update=None*)

Performs optimization

Variables

- **fun** – Objective function
- **funargs** – Keyword arguments that are passed to the function
- **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** –

__init__ (*fun, ub, lb, verbose=10, options=None, funargs=None, hessian_update=None*)
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)
- **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.

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 (*fval*, *x*, *grad*)

Check whether optimization has converged.

Parameters

- **fval** – updated objective function value
- **x** – updated optimization variables
- **grad** – updated objective function gradient

Return type `None`

check_finite (*grad=None*, *hess=None*)

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

Parameters

- **grad** (`typing.Optional[numpy.ndarray]`) – gradient to be checked for finiteness, if not provided, current one will be checked
- **hess** (`typing.Optional[numpy.ndarray]`) – Hessian (approximation) to be checked for finiteness, if not provided, current one will be checked

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 it's 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*, *fval*)

Prints diagnostic information about the current step to the log

Parameters

- **accepted** (`bool`) – flag indicating whether the current step was accepted
- **step** (`fides.trust_region.Step`) – proposal step
- **fval** (`float`) – new fval if step is accepted

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*)

Minimize the objective function the interior trust-region reflective algorithm described by [ColemanLi1994] and [ColemanLi1996] Convergence with respect to function value is achieved when

math: $|f_{k+1} - f_k| < \text{options}[fatol] - f_k$ $\text{options}[frtol]$. Similarly, convergence with respect to optimization variables is achieved when $\|x_{k+1} - x_k\| < \text{options}[xatol] - x_k$ $\text{options}[xrtol]$. Convergence with respect to the gradient is achieved when $\|g_k\| < \text{options}[gatal]$ or $\|g_k\| < \text{options}[grtol] * f_k$. Other than that, optimization can be terminated when iterations exceed $\text{options}[maxiter]$ or the elapsed time is expected to exceed $\text{options}[maxtime]$ 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 (*x_new*, *fval_new*, *grad_new*)

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

Parameters

- **x_new** (`numpy.ndarray`) –
- **fval_new** (`float`) –
- **grad_new** (`numpy.ndarray`) –

Return type `None`

Returns

update (*step*, *x_new*, *fval_new*, *grad_new*, *hess_new*=`None`)

Update self according to employed step

Parameters

- **step** (`fides.trust_region.Step`) – Employed step
- **x_new** (`numpy.ndarray`) – New optimization variable values
- **fval_new** (`float`) – Objective function value at **x_new**
- **grad_new** (`numpy.ndarray`) – Objective function gradient at **x_new**
- **hess_new** (`typing.Optional[numpy.ndarray]`) – (Approximate) objective function Hessian at **x_new**

Return type `None`

update_tr_radius (*fval*, *grad*, *step*, *dv*)

Update the trust region radius

Parameters

- **fval** (`float`) – new function value if step defined by **step_sx** is taken
- **grad** (`numpy.ndarray`) – new gradient value if step defined by **step_sx** is taken
- **step** (`fides.trust_region.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

2.3 Trust Region Steps

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

Classes Summary

<i>GradientStep</i> (x, sg, hess, scaling, ...)	This class provides the machinery to compute a gradient step.
<i>RefinedStep</i> (x, sg, hess, scaling, ...)	This class provides the machinery to refine a step based on interior point optimization
<i>Step</i> (x, sg, hess, scaling, g_dscaling, ...)	Base class for the computation of a proposal step
<i>TRStep2D</i> (x, sg, hess, scaling, g_dscaling, ...)	This class provides the machinery to compute an approximate solution of the trust region subproblem according to a 2D subproblem
<i>TRStepFull</i> (x, sg, hess, scaling, g_dscaling, ...)	This class provides the machinery to compute an exact solution of the trust region subproblem.
<i>TRStepReflected</i> (x, sg, hess, scaling, ...)	This class provides the machinery to compute a reflected step based on trust region subproblem solution that hit the boundaries.
<i>TRStepTruncated</i> (x, sg, hess, scaling, ...)	This class provides the machinery to compute a reduced step based on trust region subproblem solution that hit the boundaries.

Functions Summary

<i>normalize</i> (v)	Inplace normalization of a vector
<i>quadratic_form</i> (Q, p, x)	Computes the quadratic form $x^T Q x + x^T p$
<i>stepback_refine</i> (steps, x, sg, hess, scaling, ...)	Refine a promising subset of the provided steps based on trust-constr optimization
<i>stepback_reflect</i> (tr_step, x, sg, hess, ...)	Compute new proposal steps according to a reflection strategy.
<i>stepback_truncate</i> (tr_step, x, sg, hess, ...)	Compute new proposal steps according to a truncation strategy.
<i>trust_region</i> (x, g, hess, scaling, delta, dv, ...)	Compute a step according to the solution of the trust-region subproblem.

Classes

class fides.trust_region.**GradientStep** (x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb)

This class provides the machinery to compute a gradient step.

__init__ (x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb)

Parameters

- **x** – Reference point
- **sg** – Gradient in rescaled coordinates
- **hess** – Hessian in unscaled coordinates

- **scaling** – Matrix that defines scaling transformation
- **g_dscaling** – Unscaled gradient multiplied by derivative of scaling transformation
- **delta** – Trust region Radius in scaled coordinates
- **theta** – Stepback parameter that controls how close steps are allowed to get to the boundary
- **ub** – Upper boundary
- **lb** – Lower boundary

class fides.trust_region.**RefinedStep**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

This class provides the machinery to refine a step based on interior point optimization

__init__(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

Parameters

- **x** – Reference point
- **sg** – Gradient in rescaled coordinates
- **hess** – Hessian in unscaled coordinates
- **scaling** – Matrix that defines scaling transformation
- **g_dscaling** – Unscaled gradient multiplied by derivative of scaling transformation
- **delta** – Trust region Radius in scaled coordinates
- **theta** – Stepback parameter that controls how close steps are allowed to get to the boundary
- **ub** – Upper boundary
- **lb** – Lower boundary

calculate()

Calculates step and the expected objective function value according to the quadratic approximation

class fides.trust_region.**Step**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb*)

Base class for the computation of a proposal step

Variables

- **x** – Current state of optimization variables
- **s** – Proposed step
- **sc** – Coefficients in the 1D/2D subspace that defines the affine transformed step ss : $ss = subspace * sc$
- **ss** – Affine transformed step: $s = scaling * ss$
- **og_s** – s without step back
- **og_sc** – sc without step back
- **og_ss** – ss without step back
- **sg** – Rescaled gradient $scaling * g$
- **hess** – Hessian of the objective function at x
- **g_dscaling** – $diag(g) * dscaling$
- **delta** – Trust region radius in the transformed space defined by scaling matrix

- **theta** – Controls step back, fraction of step to take if full step would reach breakpoint
- **lb** – Lower boundaries for x
- **ub** – Upper boundaries for x
- **minbr** – Maximal fraction of step s that can be taken to reach first breakpoint
- **iminbr** – Index of x that specifies the variable that will hit the breakpoint if a step minbr * s is taken
- **qpval** – Value of the quadratic subproblem for the proposed step
- **shess** – Matrix of the full quadratic problem
- **cg** – Projection of the g_hat to the subspace
- **chess** – Projection of the B to the subspace
- **reflection_count** – Number of reflections that were applied to obtain this step
- **truncation_count** – Number of reflections that were applied to obtain this step
- **type** – Identifier that allows identification of subclasses

`__init__(x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb)`

Parameters

- **x** (`numpy.ndarray`) – Reference point
- **sg** (`numpy.ndarray`) – Gradient in rescaled coordinates
- **hess** (`numpy.ndarray`) – Hessian in unscaled coordinates
- **scaling** (`scipy.sparse.csc.csc_matrix`) – Matrix that defines scaling transformation
- **g_dscaling** (`scipy.sparse.csc.csc_matrix`) – Unscaled gradient multiplied by derivative of scaling transformation
- **delta** (`float`) – Trust region Radius in scaled coordinates
- **theta** (`float`) – Stepback parameter that controls how close steps are allowed to get to the boundary
- **ub** (`numpy.ndarray`) – Upper boundary
- **lb** (`numpy.ndarray`) – Lower boundary

`calculate()`

Calculates step and the expected objective function value according to the quadratic approximation

`compute_step()`

Compute the step as solution to the trust region subproblem. Special code is used for the special case 1-dimensional subspace case

Return type `None`

`reduce_to_subspace()`

This function projects the matrix shess and the vector sg to the subspace

Return type `None`

`step_back()`

This function truncates the step based on the distance of the current point to the boundary.

class fides.trust_region.**TRStep2D**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb*)

This class provides the machinery to compute an approximate solution of the trust region subproblem according to a 2D subproblem

__init__(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb*)

Parameters

- **x** – Reference point
- **sg** – Gradient in rescaled coordinates
- **hess** – Hessian in unscaled coordinates
- **scaling** – Matrix that defines scaling transformation
- **g_dscaling** – Unscaled gradient multiplied by derivative of scaling transformation
- **delta** – Trust region Radius in scaled coordinates
- **theta** – Stepback parameter that controls how close steps are allowed to get to the boundary
- **ub** – Upper boundary
- **lb** – Lower boundary

class fides.trust_region.**TRStepFull**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb*)

This class provides the machinery to compute an exact solution of the trust region subproblem.

__init__(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb*)

Parameters

- **x** – Reference point
- **sg** – Gradient in rescaled coordinates
- **hess** – Hessian in unscaled coordinates
- **scaling** – Matrix that defines scaling transformation
- **g_dscaling** – Unscaled gradient multiplied by derivative of scaling transformation
- **delta** – Trust region Radius in scaled coordinates
- **theta** – Stepback parameter that controls how close steps are allowed to get to the boundary
- **ub** – Upper boundary
- **lb** – Lower boundary

class fides.trust_region.**TRStepReflected**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

This class provides the machinery to compute a reflected step based on trust region subproblem solution that hit the boundaries.

__init__(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

Parameters **step** (*fides.trust_region.Step*) – Trust-region step that is reflected

class fides.trust_region.**TRStepTruncated**(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

This class provides the machinery to compute a reduced step based on trust region subproblem solution that hit the boundaries.

__init__(*x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb, step*)

Parameters `step` (`fides.trust_region.Step`) – Trust-region step that is reduced

Functions

`fides.trust_region.normalize(v)`
Inplace normalization of a vector

Parameters `v` (`numpy.ndarray`) – vector to be normalized

Return type `None`

`fides.trust_region.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.trust_region.stepback_refine(steps, x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb)`
Refine a promising subset of the provided steps based on trust-constr optimization

Parameters

- `steps` (`typing.Sequence[fides.trust_region.Step]`) – Reference trust region step that will be reflect
- `x` (`numpy.ndarray`) – Current values of the optimization variables
- `sg` (`numpy.ndarray`) – Rescaled objective function gradient at `x`
- `hess` (`numpy.ndarray`) – (Approximate) objective function Hessian at `x`
- `g_dscaling` (`scipy.sparse.csc.csc_matrix`) – Unscaled gradient multiplied by derivative of scaling transformation
- `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
- `theta` (`float`) – parameter regulating stepback
- `lb` (`numpy.ndarray`) – lower optimization variable boundaries
- `ub` (`numpy.ndarray`) – upper optimization variable boundaries

Return type `typing.List[fides.trust_region.Step]`

Returns New proposal steps

`fides.trust_region.stepback_reflect(tr_step, x, sg, hess, scaling, g_dscaling, delta, theta, ub, lb)`

Compute new proposal steps according to a reflection strategy.

Parameters

- `tr_step` (`fides.trust_region.Step`) – Reference trust region step that will be reflect

- **x** (`numpy.ndarray`) – Current values of the optimization variables
- **sg** (`numpy.ndarray`) – Rescaled objective function gradient at x
- **hess** (`numpy.ndarray`) – (Approximate) objective function Hessian at x
- **g_dscaling** (`scipy.sparse.csc.csc_matrix`) – Unscaled gradient multiplied by derivative of scaling transformation
- **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
- **theta** (`float`) – parameter regulating stepback
- **lb** (`numpy.ndarray`) – lower optimization variable boundaries
- **ub** (`numpy.ndarray`) – upper optimization variable boundaries

Return type `typing.List[fides.trust_region.Step]`

Returns New proposal steps

`fides.trust_region.stepback_truncate` (*tr_step*, *x*, *sg*, *hess*, *scaling*, *g_dscaling*, *delta*, *theta*, *ub*, *lb*)

Compute new proposal steps according to a truncation strategy.

Parameters

- **tr_step** (`fides.trust_region.Step`) – Reference trust region step that will be reflect
- **x** (`numpy.ndarray`) – Current values of the optimization variables
- **sg** (`numpy.ndarray`) – Rescaled objective function gradient at x
- **hess** (`numpy.ndarray`) – (Approximate) objective function Hessian at x
- **g_dscaling** (`scipy.sparse.csc.csc_matrix`) – Unscaled gradient multiplied by derivative of scaling transformation
- **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
- **theta** (`float`) – parameter regulating stepback
- **lb** (`numpy.ndarray`) – lower optimization variable boundaries
- **ub** (`numpy.ndarray`) – upper optimization variable boundaries

Return type `typing.List[fides.trust_region.Step]`

Returns New proposal steps

`fides.trust_region.trust_region` (*x*, *g*, *hess*, *scaling*, *delta*, *dv*, *theta*, *lb*, *ub*, *subspace_dim*, *step-back_strategy*, *refine_stepback*)

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.
- **refine_stepback** (`bool`) – If set to True, proposed steps that are computed via the specified `stepback_strategy` will be refined via optimization.

Return type `fides.trust_region.Step`

Returns `s`: proposed step, `ss`: rescaled proposed step, `qpval`: expected function value according to local quadratic approximation, `subspace`: computed subspace for reuse if proposed step is not accepted, `steptype`: type of step that was selected for proposal

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 λ , where s is the subproblem solution according to
<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 ubproblem 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.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)`

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 to 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

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>BFGS([hess_init])</code>	Broyden-Fletcher-Goldfarb-Shanno update strategy.
<code>DFP([hess_init])</code>	Davidon-Fletcher-Powell update strategy.
<code>HessianApproximation([hess_init])</code>	Abstract class from which Hessian update strategies should subclass
<code>SR1([hess_init])</code>	Symmetric Rank 1 update strategy.

Classes

class `fides.hessian_approximation.BFGS` (*hess_init=None*)

Broyden-Fletcher-Goldfarb-Shanno update strategy. This is a rank 2 update strategy that always yields positive-semidefinite hessian approximations.

class `fides.hessian_approximation.DFP` (*hess_init=None*)

Davidon-Fletcher-Powell update strategy. This is a rank 2 update strategy that always yields positive-semidefinite hessian approximations. It usually does not perform as well as the BFGS strategy, but included for the sake of completeness.

class `fides.hessian_approximation.HessianApproximation` (*hess_init=None*)

Abstract class from which Hessian update strategies should subclass

`__init__` (*hess_init=None*)

Creates Hessian update strategy instance

Parameters `hess_init` (`typing.Optional[numpy.ndarray]`) – Initial guess for the Hessian, if empty Identity matrix will be used

`get_mat` ()

Getter for the Hessian approximation :rtype: `numpy.ndarray` :return:

`init_mat` (*dim*)

Initializes this approximation instance and checks the dimensionality

Parameters `dim` (`int`) – dimension of optimization variables

class `fides.hessian_approximation.SR1` (*hess_init=None*)

Symmetric Rank 1 update strategy. This updating strategy may yield indefinite hessian approximations.

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

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.

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

FATOL = 'fatol'

absolute tolerance for convergence based on fval

FRTOL = 'frtol'

relative tolerance for convergence based on fval

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

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

MAXITER = 'maxiter'
    maximum number of allowed iterations

MAXTIME = 'maxtime'
    maximum amount of walltime in seconds

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

XATOL = 'xatol'
    absolute tolerance for convergence based on x

XRTOL = 'xrtol'
    relative 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

REFLECT = 'reflect'
    recursive reflections at boundary

SINGLE_REFLECT = 'reflect_single'
    single reflection at boundary

TRUNCATE = 'truncate'
    truncate step at boundary and resolve subproblem

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

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

TWO = '2D'
    Two dimensional Newton/Gradient subspace
```


INDICES AND TABLES

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

PYTHON MODULE INDEX

f

- `fides`, [5](#)
- `fides.constants`, [18](#)
- `fides.hessian_approximation`, [17](#)
- `fides.logging`, [18](#)
- `fides.minimize`, [5](#)
- `fides.subproblem`, [15](#)
- `fides.trust_region`, [8](#)

Symbols

`__init__()` (*fides.hessian_approximation.HessianApproximation* method), 18
`__init__()` (*fides.minimize.Optimizer* method), 6
`__init__()` (*fides.trust_region.GradientStep* method), 9
`__init__()` (*fides.trust_region.RefinedStep* method), 10
`__init__()` (*fides.trust_region.Step* method), 11
`__init__()` (*fides.trust_region.TRStep2D* method), 12
`__init__()` (*fides.trust_region.TRStepFull* method), 12
`__init__()` (*fides.trust_region.TRStepReflected* method), 12
`__init__()` (*fides.trust_region.TRStepTruncated* method), 12

B

BFGS (*class in fides.hessian_approximation*), 18

C

`calculate()` (*fides.trust_region.RefinedStep* method), 10
`calculate()` (*fides.trust_region.Step* method), 11
`check_continue()` (*fides.minimize.Optimizer* method), 6
`check_convergence()` (*fides.minimize.Optimizer* method), 6
`check_finite()` (*fides.minimize.Optimizer* method), 7
`check_in_bounds()` (*fides.minimize.Optimizer* method), 7
`compute_step()` (*fides.trust_region.Step* method), 11

D

DELTA_INIT (*fides.constants.Options* attribute), 19
 DFP (*class in fides.hessian_approximation*), 18
 DID_NOT_RUN (*fides.constants.ExitFlag* attribute), 19
 dsecular() (*in module fides.subproblem*), 16
 dslam() (*in module fides.subproblem*), 16

E

FIXED_BOUNDARY (*fides.constants.ExitFlag* attribute), 19
 ExitFlag (*class in fides.constants*), 19

F

FATOL (*fides.constants.Options* attribute), 19
 fides
 module, 5
 fides.constants
 module, 18
 fides.hessian_approximation
 module, 17
 fides.logging
 module, 18
 fides.minimize
 module, 5
 fides.subproblem
 module, 15
 fides.trust_region
 module, 8
 FRTOL (*fides.constants.Options* attribute), 19
 FTOL (*fides.constants.ExitFlag* attribute), 19
 FULL (*fides.constants.SubSpaceDim* attribute), 20

G

GATOL (*fides.constants.Options* attribute), 19
 get_affine_scaling() (*fides.minimize.Optimizer* method), 7
 get_mat() (*fides.hessian_approximation.HessianApproximation* method), 18
 GradientStep (*class in fides.trust_region*), 9
 GRTOL (*fides.constants.Options* attribute), 20
 GTOL (*fides.constants.ExitFlag* attribute), 19

H

HessianApproximation (*class in fides.hessian_approximation*), 18

I

init_mat() (*fides.hessian_approximation.HessianApproximation* method), 18

L

`log_header()` (*fides.minimize.Optimizer method*), 7
`log_step()` (*fides.minimize.Optimizer method*), 7
`log_step_initial()` (*fides.minimize.Optimizer method*), 7

M

`make_non_degenerate()`
 (*fides.minimize.Optimizer method*), 7
`MAXITER` (*fides.constants.ExitFlag attribute*), 19
`MAXITER` (*fides.constants.Options attribute*), 20
`MAXTIME` (*fides.constants.ExitFlag attribute*), 19
`MAXTIME` (*fides.constants.Options attribute*), 20
`minimize()` (*fides.minimize.Optimizer method*), 7
`MIXED` (*fides.constants.StepBackStrategy attribute*), 20
`module`
 fides, 5
 fides.constants, 18
 fides.hessian_approximation, 17
 fides.logging, 18
 fides.minimize, 5
 fides.subproblem, 15
 fides.trust_region, 8

N

`normalize()` (*in module fides.trust_region*), 13
`NOT_FINITE` (*fides.constants.ExitFlag attribute*), 19

O

`Optimizer` (*class in fides.minimize*), 5
`Options` (*class in fides.constants*), 19

Q

`quadratic_form()` (*in module fides.trust_region*), 13

R

`reduce_to_subspace()` (*fides.trust_region.Step method*), 11
`RefinedStep` (*class in fides.trust_region*), 10
`REFLECT` (*fides.constants.StepBackStrategy attribute*), 20

S

`secular()` (*in module fides.subproblem*), 16
`SINGLE_REFLECT` (*fides.constants.StepBackStrategy attribute*), 20
`slam()` (*in module fides.subproblem*), 16
`solve_1d_trust_region_subproblem()` (*in module fides.subproblem*), 17
`solve_nd_trust_region_subproblem()` (*in module fides.subproblem*), 17
`SR1` (*class in fides.hessian_approximation*), 18
`Step` (*class in fides.trust_region*), 10

`step_back()` (*fides.trust_region.Step method*), 11
`stepback_refine()` (*in module fides.trust_region*), 13
`stepback_reflect()` (*in module fides.trust_region*), 13
`STEPBACK_STRAT` (*fides.constants.Options attribute*), 20
`stepback_truncate()` (*in module fides.trust_region*), 14
`StepBackStrategy` (*class in fides.constants*), 20
`SUBSPACE_DIM` (*fides.constants.Options attribute*), 20
`SubSpaceDim` (*class in fides.constants*), 20

T

`THETA_MAX` (*fides.constants.Options attribute*), 20
`track_minimum()` (*fides.minimize.Optimizer method*), 8
`TRStep2D` (*class in fides.trust_region*), 11
`TRStepFull` (*class in fides.trust_region*), 12
`TRStepReflected` (*class in fides.trust_region*), 12
`TRStepTruncated` (*class in fides.trust_region*), 12
`TRUNCATE` (*fides.constants.StepBackStrategy attribute*), 20
`trust_region()` (*in module fides.trust_region*), 14
`TWO` (*fides.constants.SubSpaceDim attribute*), 20

U

`update()` (*fides.minimize.Optimizer method*), 8
`update_tr_radius()` (*fides.minimize.Optimizer method*), 8

X

`XATOL` (*fides.constants.Options attribute*), 20
`XRTOL` (*fides.constants.Options attribute*), 20
`XTOL` (*fides.constants.ExitFlag attribute*), 19