ellcpp Documentation

Release 0.3.6

Wai-Shing Luk

Dec 01, 2022

CONTENTS

Welcome to the EllCpp website.

Tip: The webpage you are viewing used the `html_theme` of `bootstrap` in `conf.py`.

CHAPTER

ONE

LIBRARY API

Welcome to the developer reference to Exhale Companion. The code being documented here is largely meaningless and was only created to test various corner cases e.g. nested namespaces and the like.

Note: The text you are currently reading was fed to `exhale_args` using the `afterTitleDescription` key. Full reStructuredText syntax can be used.

Tip: Sphinx / Exhale support unicode! You're `conf.py` already has it's encoding declared as `# -*- coding: utf-8 -*-` **by default**. If you want to pass Unicode strings into Exhale, simply prefix them with a `u` e.g. `u""` (of course you would actually do this because you are writing with åçćëñtß or non-English).

1.1 Page Hierarchy

1.2 Class Hierarchy

1.3 File Hierarchy

Below the hierarchies comes the full API listing.

1. The text you are currently reading is provided by `afterHierarchyDescription`.
2. The Title of the next section *just below this* normally defaults to `Full API`, but the title was changed by providing an argument to `fullApiSubSectionTitle`.
3. You can control the number of bullet points for each linked item on the remainder of the page using `fullToctreeMaxDepth`.

1.4 Custom Full API SubSection Title

1.4.1 Namespaces

Namespace algo

Page Contents

Functions

- *Template Function algo::half_nonnegative*

Namespace fun

Page Contents

Classes

- *Template Struct Fraction*

Functions

- *Template Function fun::gcd(_Mn, _Mn)*
- *Template Function fun::gcd(Mn, Mn)*
- *Template Function fun::lcm(_Mn, _Mn)*
- *Template Function fun::lcm(Mn, Mn)*
- *Template Function fun::operator**
- *Template Function fun::operator+(int&&, const Fraction<Z>&)*
- *Template Function fun::operator+(const Z&, const Fraction<Z>&)*
- *Template Function fun::operator-(const Z&, const Fraction<Z>&)*
- *Template Function fun::operator-(int&&, const Fraction<Z>&)*
- *Template Function fun::operator<<*

Namespace py

Page Contents

Classes

- *Template Struct key_iterator*
- *Template Class dict*
- *Template Class set*

Functions

- *Template Function py::enumerate*
- *Template Function py::len(const dict<Key, T>&)*
- *Template Function py::len(const set<Key>&)*
- *Template Function py::operator<(const Key&, const set<Key>&)*
- *Template Function py::operator<(const Key&, const dict<Key, T>&)*
- *Template Function py::range(T)*
- *Template Function py::range(T, T)*

Namespace xn

Page Contents

Detailed Description

View Classes provide node, edge and degree "views" of a graph. Views for nodes, edges and degree are provided for all base graph classes. A view means a read-only object that is quick to create, automatically updated when the graph changes, and provides basic access like `n : V`, `for n : V`, `V[n]` and sometimes set operations. The views are read-only iterable containers that are updated as the graph is updated. As with dicts, the graph should not be updated while (iterating through the view. Views can be iterated multiple times. Edge and Node views also allow data attribute lookup. The resulting attribute dict is writable as `G.edges[3, 4]["color"]="red"` Degree views allow lookup of degree values for single nodes. Weighted degree is supported with the `weight` argument. *Template Class NodeView*

V = G.nodes (or *V = G.nodes()*) allows *len(V)*, *n : V*, set operations e.g. "G.nodes & H.nodes", and *dd = G.nodes[n]*, where *dd* is the node data dict. Iteration is over the nodes by default.

NodeDataView

To iterate over (node, data) pairs, use arguments to *G.nodes()* to create a DataView e.g. *DV = G.nodes(data="color", default="red")*. The DataView iterates as *for n, color : DV* and allows *(n, "red") : DV*. Using *DV = G.nodes(data=true)*, the DataViews use the full datadict : writeable form also allowing contain testing as *(n, {"color": "red"}) : VD*. DataViews allow set operations when data attributes are hashable.

DegreeView

V = G.degree allows iteration over (node, degree) pairs as well as lookup: *deg=V[n]*. There are many flavors of DegreeView for In/Out/Directed/Multi. For Directed Graphs, *G.degree* counts both : and out going edges. *G.out_degree* && *G.in_degree* count only specific directions. Weighted degree using edge data attributes is provide via *V = G.degree(weight="attr_name")* where any string with the attribute name can be used. *weight=None* is the default. No set operations are implemented for degrees, use NodeView.

The argument *nbunch* restricts iteration to nodes : nbunch. The DegreeView can still lookup any node even if (nbunch is specified.

Template Class EdgeView

V = G.edges or *V = G.edges()* allows iteration over edges as well as *e : V*, set operations and edge data lookup *dd = G.edges[2, 3]*. Iteration is over 2-tuples *(u, v)* for Graph/DiGraph. For multigraphs edges 3-tuples *(u, v, key)* are the default but 2-tuples can be obtained via *V = G.edges(keys=false)*.

Set operations for directed graphs treat the edges as a set of 2-tuples. For undirected graphs, 2-tuples are not a unique representation of edges. So long as the set being compared to contains unique representations of its edges, the set operations will act as expected. If the other set contains both *(0, 1)* and *(1, 0)* however, the result of set operations may contain both representations of the same edge.

EdgeDataView

Edge data can be reported using an EdgeDataView typically created by calling an EdgeView: *DV = G.edges(data="weight", default=1)*. The EdgeDataView allows iteration over edge tuples, membership checking but no set operations.

Iteration depends on *data* and *default* and for multigraph *keys* If *data == false* (the default) then iterate over 2-tuples *(u, v)*. If *data is true* iterate over 3-tuples *(u, v, datadict)*. Otherwise iterate over *(u, v, datadict.get(data, default))*. For Multigraphs, if (*keys is true*, replace *u, v* with *u, v, key* to create 3-tuples and 4-tuples.

The argument *nbunch* restricts edges to those incident to nodes : nbunch. Exceptions Base exceptions and errors for XNetwork.

Classes

- *Struct AmbiguousSolution*
- *Struct ExceededMaxIterations*
- *Struct HasACycle*
- *Struct NodeNotFound*
- *Struct object*
- *Struct XNetworkAlgorithmError*
- *Struct XNetworkError*
- *Struct XNetworkException*
- *Struct XNetworkNoCycle*
- *Struct XNetworkNoPath*
- *Struct XNetworkNotImplemented*
- *Struct XNetworkPointlessConcept*
- *Struct XNetworkUnbounded*
- *Struct XNetworkUnfeasible*
- *Template Class AtlasView*
- *Template Class DiGraphS*
- *Template Class EdgeView*
- *Template Class grAdaptor*
- *Template Class Graph*
- *Template Class NodeView*
- *Template Class VertexView*

Typedefs

- *Typedef xn::SimpleDiGraphS*
- *Typedef xn::SimpleGraph*

Variables

- *Variable xn::__slots__*

1.4.2 Classes and Structs

Struct CInfo

- Defined in file_ellcpp_cut_config.hpp

Page Contents

Struct Documentation

struct **CInfo**

CInfo.

Public Members

bool **feasible**

size_t **num_iters**

CUTStatus **status**

Template Struct Fraction

- Defined in file_py2cpp_fractions-new.hpp

Page Contents

Inheritance Relationships

Base Type

- `public boost::totally_ordered< Fraction< Z >, boost::totally_ordered2< Fraction< Z >, Z, boost::multipliable2< Fraction< Z >, Z, boost::dividable2< Fraction< Z >, Z > > > >`

Template Parameter Order

1. `typename Z`

Struct Documentation

template<typename **Z**>

struct **Fraction** : public boost::totally_ordered<*Fraction*<*Z*>, boost::totally_ordered2<*Fraction*<*Z*>, *Z*, boost::multipliable2<*Fraction*<*Z*>, *Z*, boost::dividable2<*Fraction*<*Z*>, *Z*>>>>

Public Types

using **_Self** = *Fraction*<*Z*>

Public Functions

inline constexpr **Fraction**(const *Z* &numerator, const *Z* &denominator)

Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline explicit constexpr **Fraction**(const *Z* &numerator)

Construct a new *Fraction* object.

Parameters **numerator** – **[in]**

constexpr **Fraction**() = default

Construct a new *Fraction* object.

inline constexpr const *Z* &**numerator**() const

Returns const Z&

inline constexpr const *Z* &**denominator**() const

Returns const Z&

inline constexpr *_Self* **abs**() const

Returns _Self

inline constexpr void **reciprocal**()

inline constexpr *_Self* **operator-**() const

Returns _Self

inline constexpr *_Self* **operator+**(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-**(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator***(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/**(*_Self* frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator***(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator*=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator*=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

template<typename **U**>
inline constexpr auto **cmp**(const *Fraction*<*U*> &frac) const

Three way comparison.

Parameters **frac** – **[in]**

Returns auto

template<typename **U**>
inline constexpr bool **operator==**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator!=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator<**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>

inline constexpr bool **operator>**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator<=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator>=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

inline constexpr auto **cmp**(const *Z* &c) const

Parameters **c** – **[in]**

Returns auto

inline constexpr bool **operator==**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator!=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator<**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator>**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator<=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator>=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline explicit constexpr **operator double**()

Returns double

inline constexpr **Fraction**(*Z* &&numerator, *Z* &&denominator) noexcept

Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline constexpr **Fraction**(const *Z* &numerator, const *Z* &denominator)

Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline constexpr void **normalize**()

inline explicit constexpr **Fraction**(*Z* &&numerator) noexcept

Construct a new *Fraction* object.

Parameters **numerator** – **[in]**

inline explicit constexpr **Fraction**(const *Z* &numerator)

Construct a new *Fraction* object.

Parameters **numerator** – **[in]**

inline constexpr auto **numerator**() const -> const *Z*&

Returns const Z&

inline constexpr auto **denominator**() const -> const *Z*&

Returns const Z&

inline constexpr auto **abs**() const -> *Fraction*

Returns *Fraction*

inline constexpr void **reciprocal**()

inline constexpr auto **operator-**() const -> *Fraction*

Returns *Fraction*

inline constexpr auto **operator+**(const *Fraction* &frac) const -> *Fraction*

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator-**(const *Fraction* &frac) const -> *Fraction*

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator***(const *Fraction* &frac) const -> *Fraction*

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator/**(*Fraction* frac) const -> *Fraction*

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator+**(const *Z* &i) const -> *Fraction*

Parameters **i** – **[in]**

Returns *Fraction*

inline constexpr auto **operator-**(const *Z* &i) const -> *Fraction*

Parameters **i** – **[in]**

Returns *Fraction*

inline constexpr auto **operator+=**(const *Fraction* &frac) -> *Fraction*&

Parameters

- **i** – **[in]**
- **i** – **[in]**
- **frac** – **[in]**

Returns *Fraction*

Returns *Fraction*

Returns *Fraction*

inline constexpr auto **operator-=**(const *Fraction* &frac) -> *Fraction*&

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator*=**(const *Fraction* &frac) -> *Fraction*&

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator/=**(const *Fraction* &frac) -> *Fraction*&

Parameters **frac** – **[in]**

Returns *Fraction*

inline constexpr auto **operator+=**(const *Z* &i) -> *Fraction*&

Parameters **i** – **[in]**

Returns *Fraction*

inline constexpr auto **operator-=**(const *Z* &i) -> *Fraction*&

Parameters **i** – **[in]**

Returns *Fraction*

inline constexpr auto **operator*=**(const *Z* &i) -> *Fraction*&

Parameters **i** – **[in]**

Returns *Fraction*

inline constexpr auto **operator/=**(const *Z* &i) -> *Fraction*&

Parameters **i** – **[in]**

Returns *Fraction*

template<typename **U**>
inline constexpr auto **cmp**(const *Fraction*<*U*> &frac) const

Three way comparison.

Parameters **frac** – **[in]**

Returns auto

inline constexpr auto **operator==**(const *Fraction*<*Z*> &rhs) const -> bool

inline constexpr auto **operator<**(const *Fraction*<*Z*> &rhs) const -> bool

inline constexpr auto **operator==**(const *Z* &rhs) const -> bool

inline constexpr auto **operator<**(const *Z* &rhs) const -> bool

inline constexpr auto **operator>**(const *Z* &rhs) const -> bool

Public Members

Z **_numerator**

Z **_denominator**

Friends

inline friend constexpr *_Self* **operator+**(const *Z* &c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend constexpr *_Self* **operator-**(const *Z* &c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend constexpr *_Self* **operator***(const *Z* &c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend constexpr *_Self* **operator+**(int &&c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**
- **c** – **[in]**
- **frac** – **[in]**
- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Returns Fraction<Z>

Returns Fraction<Z>

inline friend constexpr *_Self* **operator-**(int &&c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend constexpr *_Self* **operator***(int &&c, const *_Self* &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

template<typename **_Stream**>
inline friend *_Stream* &**operator<<**(*_Stream* &os, const *_Self* &frac)

Template Parameters

- **_Stream** –
- **Z** –

Parameters

- **os** – **[in]**
- **frac** – **[in]**

Returns _Stream&

Template Struct Info4EM

- Defined in file_ellip_ellipsoid.hpp

Page Contents

Template Parameter Order

1. `class Vec`

Struct Documentation

template<class **Vec**>

struct **Info4EM**

— (Generalized) bisection method for solving convex minimization problem P:

```
minimize     fct_0(x)
subject to   fct_j(x) <= 0
  where fct_0(x) and fct_j(x)'s are convex
```

Input: E(x) initial enclosing region max_it maximum number of iterations tol error tolerance P Representation of convex minimization problem

Requirement of P: void P.assess(x) assessment of x bool P.is_feasible() return true if x is feasible double P.f_value() returns fct_j(x) if x is infeasible for some j fct::Vec P.subgradient() returns subgradient of fct_0 if x is feasible subgradient of fct_j if x is infeasible for some j Requirement of E:

output x optimal solution status FOUND = solution found to tolerance EXCEEDMAXITER = no convergence given max_it NOTFOUND = no feasible sol'n

Public Members

bool **_is_feasible**

Vec **_g**

if x is feasible

double **_f**

subgradient at x

Vec **_x**

function value at x

Struct micp1

- Defined in file_ellip_micp_test1.cpp

Page Contents

Struct Documentation

struct **micp1**

Problem 1:

Public Types

using **Vec** = std::valarray<double>

Public Functions

inline *Info4EM*<*Vec*> **operator()**(const *Vec* &x0)

Struct Options

- Defined in file_ellcpp_cut_config.hpp

Page Contents

Struct Documentation

struct **Options**

Options.

Public Members

unsigned int **max_it** = 2000

maximum number of iterations

double **tol** = 1e-8

error tolerance

Template Struct key_iterator

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- `public Iter`

Template Parameter Order

1. `typename Iter`

Struct Documentation

template<typename **Iter**>

struct **key_iterator** : public *Iter*

Template Deduction Guide.

Template Parameters **Key** –

Public Functions

inline explicit **key_iterator**(*Iter* it)

inline auto **operator***() const

inline auto **operator++**() -> *key_iterator*&

Struct AmbiguousSolution

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **AmbiguousSolution** : public xn::*XNetworkException*

Raised if (more than one valid solution exists for an intermediary step of an algorithm.

In the face of ambiguity, refuse the temptation to guess. This may occur, for example, when trying to determine the bipartite node sets in a disconnected bipartite graph when computing bipartite matchings.

Public Functions

inline explicit **AmbiguousSolution**(std::string_view msg)

Struct ExceededMaxIterations

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **ExceededMaxIterations** : public xn::*XNetworkException*

Raised if (a loop iterates too many times without breaking. This may occur, for example, in an algorithm that computes progressively better approximations to a value but exceeds an iteration bound specified by the user.

Public Functions

inline explicit **ExceededMaxIterations**(std::string_view msg)

Struct HasACycle

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **`HasACycle`** : public xn::*XNetworkException*

Raised if (a graph has a cycle when an algorithm expects that it will have no cycles.

Public Functions

inline explicit **`HasACycle`**(std::string_view msg)

Struct NodeNotFound

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **`NodeNotFound`** : public xn::*XNetworkException*

Exception raised if (requested node is not present : the graph

Public Functions

inline explicit **`NodeNotFound`**(std::string_view msg)

Struct object

- Defined in file_xnetwork_classes_graph.hpp

Page Contents

Inheritance Relationships

Base Type

- `public py::dict< const char *, boost::any >` (*Template Class dict*)

Derived Types

- `public xn::Graph< nodeview_t, adjlist_t >` (*Template Class Graph*)
- `public xn::Graph< __nodeview_t, adjlist_t >` (*Template Class Graph*)

Struct Documentation

struct **object** : public py::*dict*<const char*, boost::any>

Base class for undirected graphs.

A *Graph* stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) C++ objects with optional key/value attributes. By convention `None` is not used as a node.

Edges are represented as links between nodes with optional key/value attributes.

Parameters

node_container : input graph (optional, default: None) Data to initialize graph. If None (default) an empty graph is created. The data can be any format that is supported by the to_networkx_graph() function, currently including edge list, dict of dicts, dict of lists, NetworkX graph, NumPy matrix or 2d ndarray, SciPy sparse matrix, or PyGraphviz graph.

See Also

DiGraph MultiGraph MultiDiGraph OrderedGraph

Examples

Create an empty graph structure (a "null graph") with 5 nodes and no edges.

auto v = std::vector{3, 4, 2, 8}; auto G = *xn::Graph(v)*;

auto va = *py::dict*{{3, 0.1}, {4, 0.5}, {2, 0.2}}; auto G = *xn::Graph(va)*;

auto r = py::range(100); auto G = *xn::Graph(r)*;

G can be grown in several ways.

Nodes:**

Add one node at a time:

G.add_node(1)

Add the nodes from any container (a list, dict, set or even the lines from a file or the nodes from another graph).

```
G.add_nodes_from([2, 3]) G.add_nodes_from(range(100, 110)) H = xn::path_graph(10) G.add_nodes_from(H)
```

In addition to strings and integers any hashable C++ object (except None) can represent a node, e.g. a customized node object, or even another *Graph*.

```
G.add_node(H)
```

Edges:**

G can also be grown by adding edges.

Add one edge,

```
G.add_edge(1, 2);
```

a list of edges,

```
G.add_edges_from([(1, 2), (1, 3)]);
```

or a collection of edges,

```
G.add_edges_from(H.edges());
```

If some edges connect nodes not yet in the graph, the nodes are added automatically. There are no errors when adding nodes or edges that already exist.

Attributes:**

Each graph can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using direct manipulation of the attribute dictionaries named graph, node and edge respectively.

G.graph["day"] = boost::any("Friday");

{'day': 'Friday'}

Subclasses (Advanced):**

The *Graph* class uses a container-of-container-of-container data structure. The outer dict (node_dict) holds adjacency information keyed by node. The next dict (adjlist_dict) represents the adjacency information and holds edge data keyed by neighbor. The inner dict (edge_attr_dict) represents the edge data and holds edge attribute values keyed by attribute names.

Each of these three dicts can be replaced in a subclass by a user defined dict-like object. In general, the dict-like features should be maintained but extra features can be added. To replace one of the dicts create a new graph class by changing the class(!) variable holding the factory for that dict-like structure. The variable names are node_dict_factory, node_attr_dict_factory, adjlist_inner_dict_factory, adjlist_outer_dict_factory, edge_attr_dict_factory and graph_attr_dict_factory.

node_dict_factory : function, (default: dict) Factory function to be used to create the dict containing node attributes, keyed by node id. It should require no arguments and return a dict-like object

node_attr_dict_factory: function, (default: dict) Factory function to be used to create the node attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object

adjlist_outer_dict_factory : function, (default: dict) Factory function to be used to create the outer-most dict in the data structure that holds adjacency info keyed by node. It should require no arguments and return a dict-like object.

adjlist_inner_dict_factory : function, (default: dict) Factory function to be used to create the adjacency list dict which holds edge data keyed by neighbor. It should require no arguments and return a dict-like object

edge_attr_dict_factory : function, (default: dict) Factory function to be used to create the edge attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

graph_attr_dict_factory : function, (default: dict) Factory function to be used to create the graph attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

Typically, if your extension doesn't impact the data structure all methods will inherit without issue except: `to_directed/to_undirected`. By default these methods create a DiGraph/Graph class and you probably want them to create your extension of a DiGraph/Graph. To facilitate this we define two class variables that you can set in your subclass.

to_directed_class : callable, (default: DiGraph or MultiDiGraph) Class to create a new graph structure in the `to_directed` method. If `None`, a NetworkX class (DiGraph or MultiDiGraph) is used.

to_undirected_class : callable, (default: *Graph* or MultiGraph) Class to create a new graph structure in the `to_undirected` method. If `None`, a NetworkX class (*Graph* or MultiGraph) is used.

Examples

Create a low memory graph class that effectively disallows edge attributes by using a single attribute dict for all edges. This reduces the memory used, but you lose edge attributes.

class ThinGraph(xn::Graph):

... all_edge_dict = {'weight': 1} ... def single_edge_dict(self): ... return self.all_edge_dict ... edge_attr_dict_factory = single_edge_dict

G = ThinGraph() G.add_edge(2, 1) G[2][1]

{'weight': 1}

G.add_edge(2, 2) G[2][1] is G[2][2]

True

Please see :mod:~`networkx.classes.ordered` for more examples of creating graph subclasses by overwriting the base class `dict` with a dictionary-like object.

Subclassed by *xn::Graph< nodeview_t, adjlist_t >*, *xn::Graph< __nodeview_t, adjlist_t >*

Struct XNetworkAlgorithmError

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Derived Types

- `public xn::XNetworkUnbounded` (*Struct XNetworkUnbounded*)
- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **XNetworkAlgorithmError** : public xn::*XNetworkException*

Exception for unexpected termination of algorithms.

Subclassed by *xn::XNetworkUnbounded*, *xn::XNetworkUnfeasible*

Public Functions

inline explicit **XNetworkAlgorithmError**(std::string_view msg)

Struct XNetworkError

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **XNetworkError** : public xn::*XNetworkException*

Exception for a serious error : XNetwork

Public Functions

inline explicit **XNetworkError**(std::string_view msg)

Struct XNetworkException

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public runtime_error`

Derived Types

- `public xn::AmbiguousSolution` (*Struct AmbiguousSolution*)
- `public xn::ExceededMaxIterations` (*Struct ExceededMaxIterations*)
- `public xn::HasACycle` (*Struct HasACycle*)
- `public xn::NodeNotFound` (*Struct NodeNotFound*)
- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)
- `public xn::XNetworkError` (*Struct XNetworkError*)
- `public xn::XNetworkNotImplemented` (*Struct XNetworkNotImplemented*)
- `public xn::XNetworkPointlessConcept` (*Struct XNetworkPointlessConcept*)

Struct Documentation

struct **`XNetworkException`** : public runtime_error

Base class for exceptions : XNetwork.

Subclassed by *xn::AmbiguousSolution*, *xn::ExceededMaxIterations*, *xn::HasACycle*, *xn::NodeNotFound*, *xn::XNetworkAlgorithmError*, *xn::XNetworkError*, *xn::XNetworkNotImplemented*, *xn::XNetworkPointlessConcept*

Public Functions

inline explicit **XNetworkException**(std::string_view msg)

Struct XNetworkNoCycle

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **XNetworkNoCycle** : public xn::*XNetworkUnfeasible*

Exception for algorithms that should return a cycle when running on graphs where such a cycle does not exist.

Public Functions

inline explicit **XNetworkNoCycle**(std::string_view msg)

Struct XNetworkNoPath

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **`XNetworkNoPath`** : public xn::*XNetworkUnfeasible*

Exception for algorithms that should return a path when running on graphs where such a path does not exist.

Public Functions

inline explicit **`XNetworkNoPath`**(std::string_view msg)

Struct XNetworkNotImplemented

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **`XNetworkNotImplemented`** : public xn::*XNetworkException*

Exception raised by algorithms not implemented for a type of graph.

Public Functions

inline explicit **XNetworkNotImplemented**(std::string_view msg)

Struct XNetworkPointlessConcept

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **XNetworkPointlessConcept** : public xn::*XNetworkException*

Raised when a null graph is provided as input to an algorithm that cannot use it.

The null graph is sometimes considered a pointless concept [1]_, thus the name of the exception.

References

.. [1] Harary, F. and Read, R. "Is the Null Graph a Pointless

Concept?" In Graphs and Combinatorics Conference, George Washington University. New York: Springer-Verlag, 1973.

Public Functions

inline explicit **XNetworkPointlessConcept**(std::string_view msg)

Struct XNetworkUnbounded

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)

Struct Documentation

struct **XNetworkUnbounded** : public xn::*XNetworkAlgorithmError*

Exception raised by algorithms trying to solve a maximization or a minimization problem instance that is unbounded.

Public Functions

inline explicit **XNetworkUnbounded**(std::string_view msg)

Struct XNetworkUnfeasible

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)

Derived Types

- `public xn::XNetworkNoCycle` (*Struct XNetworkNoCycle*)
- `public xn::XNetworkNoPath` (*Struct XNetworkNoPath*)

Struct Documentation

struct **`XNetworkUnfeasible`** : public xn::*XNetworkAlgorithmError*

Exception raised by algorithms trying to solve a problem instance that has no feasible solution.

Subclassed by *xn::XNetworkNoCycle*, *xn::XNetworkNoPath*

Public Functions

inline explicit **`XNetworkUnfeasible`**(std::string_view msg)

Template Class AdjacencyView

- Defined in file_xnetwork_classes_coreviews.hpp

Page Contents

Inheritance Relationships

Base Type

- `public AtlasView< Atlas >` (*Template Class AtlasView*)

Template Parameter Order

1. `typename Atlas`

Class Documentation

template<typename **Atlas**>

class **AdjacencyView** : public *AtlasView*<*Atlas*>

An *AdjacencyView* is a Read-only Map of Maps of Maps.

It is a View into a dict-of-dict-of-dict data structure. The inner level of dict is read-write. But the outer levels are read-only.

See Also

AtlasView - View into dict-of-dict MultiAdjacencyView - View into dict-of-dict-of-dict-of-dict

Public Functions

inline explicit **AdjacencyView**(*Atlas* &d)

Template Class AtlasView

- Defined in file_xnetwork_classes_coreviews.hpp

Page Contents

Inheritance Relationships

Derived Type

- `public AdjacencyView< Atlas >` (*Template Class AdjacencyView*)

Template Parameter Order

1. `typename Atlas`

Class Documentation

template<typename **Atlas**>

class **AtlasView**

An *AtlasView* is a Read-only Mapping of Mappings.

It is a View into a dict-of-dict data structure. The inner level of dict is read-write. But the outer level is read-only.

See Also

AdjacencyView - View into dict-of-dict-of-dict MultiAdjacencyView - View into dict-of-dict-of-dict-of-dict

Interface: Mapping

Subclassed by *AdjacencyView< Atlas >*

Public Functions

inline explicit **AtlasView**(*Atlas* &d)

inline auto **size**() const

inline auto **begin**() const

inline auto **end**() const

template<typename **T**>
inline const auto &**operator[]**(const *T* &key) const

template<typename **T**>
inline auto &**operator[]**(const *T* &key)

Public Members

Atlas &**_atlas**

Template Class bsearch_adaptor

- Defined in file_ellcpp_cutting_plane.hpp

Page Contents

Template Parameter Order

1. `typename Oracle`
2. `typename Space`

Class Documentation

template<typename **Oracle**, typename **Space**>

class **bsearch_adaptor**

Template Parameters

- **Oracle** –
- **Space** –

Public Functions

inline **bsearch_adaptor**(*Oracle* &P, *Space* &S)

Construct a new bsearch adaptor object.

Parameters

- **P** – **[inout]** perform assessment on x0
- **S** – **[inout]** search Space containing x*

inline **bsearch_adaptor**(*Oracle* &P, *Space* &S, const *Options* &options)

Construct a new bsearch adaptor object.

Parameters

- **P** – **[inout]** perform assessment on x0
- **S** – **[inout]** search Space containing x*
- **options** – **[in]** maximum iteration and error tolerance etc.

inline auto **x_best**() const

get best x

Returns auto

template<typename **opt_type**>
inline auto **operator()**(const *opt_type* &t) -> bool

Parameters **t** – **[in]** the best-so-far optimal value

Returns bool

Template Class cycle_ratio_oracle

- Defined in file_ellcpp_oracles_cycle_ratio_oracle.hpp

Page Contents

Nested Relationships

Nested Types

- *Class cycle_ratio_oracle::Ratio*

Template Parameter Order

1. `typename Graph`
2. `typename Container`
3. `typename Fn1`
4. `typename Fn2`

Class Documentation

template<typename **Graph**, typename **Container**, typename **Fn1**, typename **Fn2**>

class **cycle_ratio_oracle**

Oracle for minimum ratio cycle problem.

This example solves the following convex problem:

```
max     t
s.t.    u[j] - u[i] \le mij - sij * x,
        t \le x
```

where sij is not necessarily positive.

Template Parameters

- **Graph** –
- **Container** –
- **Fn1** –
- **Fn2** –

Public Functions

inline **cycle_ratio_oracle**(const *Graph* &G, *Container* &u, *Fn1* get_cost, *Fn2* get_time)

Construct a new cycle_ratio oracle object.

Parameters

- **G** – **[in]**
- **u** – **[inout]**
- **get_cost** – **[in]**

cycle_ratio_oracle(const *cycle_ratio_oracle*&) = delete

cycle_ratio_oracle &**operator=**(const *cycle_ratio_oracle*&) = delete

cycle_ratio_oracle(*cycle_ratio_oracle*&&) = default

inline auto **operator()**(const double &x, double &t) -> std::tuple<Cut, bool>

Make object callable for cutting_plane_dc()

See also:

cutting_plane_dc

Parameters

- **x** – **[in]**
- **t** – **[in]** the best-so-far optimal value

Returns std::tuple<Cut, double>

Class cycle_ratio_oracle::Ratio

- Defined in file_ellcpp_oracles_cycle_ratio_oracle.hpp

Page Contents

Nested Relationships

This class is a nested type of *Template Class cycle_ratio_oracle*.

Class Documentation

class **Ratio**

Ratio.

Public Functions

inline **Ratio**(const Graph &G, Fn1 get_cost, Fn2 get_time)

Construct a new Ratio object.

Parameters

- **G** – **[in]**
- **get_cost** – **[in]**
- **get_time** – **[in]**

inline auto **eval**(const edge_t &e, const double &x) const -> double

Evaluate function.

Parameters

- **e** – **[in]**
- **x** – **[in]** (,) in log scale

Returns double

inline auto **grad**(const edge_t &e, const double &x) const -> double

Gradient function.

Parameters

- **e** – **[in]**
- **x** – **[in]** (,) in log scale

Returns double

Class ell

- Defined in file_ellcpp_ell.hpp

Page Contents

Inheritance Relationships

Derived Type

- `public ell_stable` (*Class ell_stable*)

Class Documentation

class **ell**

Ellipsoid Search Space.

ell = {x | (x - xc)' Q^-1 (x - xc) }

Keep Q symmetric but no promise of positive definite

Subclassed by *ell_stable*

Public Types

using **Arr** = xt::xarray<double, xt::layout_type::row_major>

Public Functions

inline **ell**(const *Arr* &val, *Arr* x) noexcept

Construct a new ell object.

Parameters

- **val** – **[in]**
- **x** – **[in]**

inline **ell**(const double &alpha, *Arr* x) noexcept

Construct a new ell object.

Parameters

- **alpha** – **[in]**
- **x** – **[in]**

ell(*ell* &&E) = default

Construct a new ell object.

Parameters **E** – **[in]** (move)

inline **~ell**()

Destroy the ell object.

explicit **ell**(const *ell* &E) = default

Construct a new ell object.

To avoid accidentally copying, only explicit copy is allowed

Parameters **E** –

inline auto **copy**() const -> *ell*

explicitly copy

Returns ell

inline auto **xc**() const -> *Arr*

copy the whole array anyway

Returns Arr

inline void **set_xc**(const *Arr* &xc)

Set the xc object.

Parameters **xc** – **[in]**

template<typename **T**>
auto **update**(const std::tuple<*Arr*, *T*> &cut) -> std::tuple<*CUTStatus*, double>

Update ellipsoid core function using the cut(s)

Template Parameters **T** –

Parameters **cut** – **[in]** cutting-plane

Returns std::tuple<int, double>

Public Members

bool **use_parallel_cut** = true

bool **no_defer_trick** = false

Protected Functions

auto **operator=**(const *ell* &E) -> *ell*& = delete

Construct a new ell object.

Parameters **E** – **[in]**

template<typename **V**, typename **U**>
inline **ell**(*V* &&kappa, *Arr* &&Q, *U* &&x) noexcept

Construct a new ell object.

Parameters

- **val** – **[in]**
- **x** – **[in]**

inline auto **_update_cut**(const double &beta) -> *CUTStatus*

inline *CUTStatus* **_update_cut**(const *Arr* &beta)

auto **_calc_ll_core**(const double &b0, const double &b1) -> *CUTStatus*

Calculate new ellipsoid under Parallel Cut.

g' (x - xc) + beta0 0 g' (x - xc) + beta1 0

Parameters

- **b0** – **[in]**
- **b1** – **[in]**

Returns int

auto **_calc_ll_cc**(const double &b1, const double &b1sq) -> void

Calculate new ellipsoid under Parallel Cut, one of them is central.

g' (x - xc) 0 g' (x - xc) + beta1 0

Parameters

- **b1** – **[in]**
- **b1sq** – **[in]**

auto **_calc_dc**(const double &beta) noexcept -> *CUTStatus*

Calculate new ellipsoid under Deep Cut.

g' (x - xc) + beta 0

Parameters **beta** – **[in]**

auto **_calc_cc**(const double &tau) noexcept -> void

Calculate new ellipsoid under Central Cut.

g' (x - xc) 0

Parameters **tau** – **[in]**

Protected Attributes

double **_mu** = { }

double **_rho** = { }

double **_sigma** = { }

double **_delta** = { }

double **_tsq** = { }

const int **_n**

const double **_nFloat**

const double **_nPlus1**

const double **_nMinus1**

const double **_halfN**

const double **_halfNplus1**

const double **_halfNminus1**

const double **_nSq**

const double **_c1**

const double **_c2**

const double **_c3**

double **_kappa**

Arr **_Q**

Arr **_xc**

Class ell1d

- Defined in file_ellcpp_ell1d.hpp

Page Contents

Class Documentation

class **ell1d**

Ellipsoid Method for special 1D case.

Public Types

using **return_t** = std::tuple<*CUTStatus*, double>

Public Functions

inline **ell1d**(const double &l, const double &u) noexcept

Construct a new *ell1d* object.

Parameters

- **l** – **[in]**
- **u** – **[in]**

explicit **ell1d**(const *ell1d* &E) = default

Construct a new *ell1d* object.

Parameters **E** – **[in]**

inline auto **xc**() const noexcept -> double

Returns double

inline auto **set_xc**(const double &xc) noexcept -> void

Set the xc object.

Parameters **xc** – **[in]**

auto **update**(const std::tuple<double, double> &cut) noexcept -> *return_t*

Parameters **cut** – **[in]**

Returns return_t

Class ell_stable

- Defined in file_ellcpp_ell_stable.hpp

Page Contents

Inheritance Relationships

Base Type

- `public ell` (*Class ell*)

Class Documentation

class **ell_stable** : public *ell*

Ellipsoid Search Space.

ell_stable = {x | (x - xc)' M^-1 (x - xc) } = {x | (x - xc)' L D^-1 L' (x - xc) }

Store M in the form of Lg \ D^-1 \ L' in an n x n array `Q`, and hence keep M symmetric positive definite. More stable but slightly more computation.

Public Types

using **Arr** = xt::xarray<double, xt::layout_type::row_major>

Public Functions

inline **ell_stable**(const *Arr* &val, *Arr* x) noexcept

Construct a new *ell_stable* object.

Parameters

- **val** – **[in]**
- **x** – **[in]**

inline **ell_stable**(const double &alpha, *Arr* x) noexcept

Construct a new *ell_stable* object.

Parameters

- **alpha** – **[in]**
- **x** – **[in]**

ell_stable(*ell_stable* &&E) = default

Construct a new *ell_stable* object.

Parameters **E** – **[in]** (move)

explicit **ell_stable**(const *ell_stable* &E) = default

Construct a new *ell_stable* object.

To avoid accidentally copying, only explicit copy is allowed

Parameters **E** –

inline **~ell_stable**()

Destroy the ell stable object.

inline auto **copy**() const -> *ell_stable*

explicitly copy

Returns *ell_stable*

template<typename **T**>

auto **update**(const std::tuple<*Arr*, *T*> &cut) -> std::tuple<*CUTStatus*, double>

Update ellipsoid core function using the cut(s)

Overwrite the base class. Store Q^-1 in the form of LDLT decomposition, and hence guarantee Q is symmetric positive definite.

Template Parameters **T** –

Parameters **cut** – **[in]** cutting-plane

Returns std::tuple<int, double>

Class ellipsoid

- Defined in file_ellip_ellipsoid.hpp

Page Contents

Class Documentation

class **ellipsoid**

Enclosing ellipsoid

Public Functions

inline **ellipsoid**(*Vec* &x, double rho)

Constructor

inline **ellipsoid**(*Vec* &x, const *Vec* &r)

Constructor

inline **~ellipsoid**()

inline *Vec* &**x**()

void **update**(const *Vec* &g)

CUTSTATUS **update**(const *Vec* &g, double beta)

Template Class gp_base

- Defined in file_ellip_gp_solve.hpp

Page Contents

Inheritance Relationships

Derived Type

- `public profit_max< _Tp >` (*Template Class profit_max*)

Template Parameter Order

1. `typename _Tp`

Class Documentation

template<typename **_Tp**>

class **gp_base**

Subclassed by *profit_max< _Tp >*

Public Functions

inline **gp_base**()

inline ~**gp_base**()

Info4EM<Vec> **operator()**(const Vec &x) const

template<>
Info4EM<Vec> **operator()**(const Vec &x) const

template<>
Info4EM<Vec> **operator()**(const Vec &x) const

Protected Attributes

std::vector<*posynomial*<*_Tp*>> **_M**

Class ldlt_ext

- Defined in file_ellcpp_oracles_ldlt_ext.hpp

Page Contents

Class Documentation

class **ldlt_ext**

LDLT factorization for LMI.

- LDL^T square-root-free version
- Option allow semidefinite
- A matrix A in R^{m x m} is positive definite iff v' A v > 0 for all v in R^n.
- O(p^2) per iteration, independent of N

Public Functions

inline explicit **ldlt_ext**(size_t N)

Construct a new ldlt ext object.

Parameters **N** – **[in]** dimension

ldlt_ext(const *ldlt_ext*&) = delete

ldlt_ext &**operator=**(const *ldlt_ext*&) = delete

ldlt_ext(*ldlt_ext*&&) = default

inline auto **factorize**(const Mat &A) -> bool

Perform LDLT Factorization.

If A is positive definite, then p is zero. If it is not, then p is a positive integer, such that $v = R^-1 e_p$ is a certificate vector to make $v'*A[:p,:p]*v < 0$

Parameters **A** – **[in]** Symmetric Matrix

template<typename **Callable**, bool **Allow_semidefinite** = false>
inline auto **factor**(*Callable* &&getA) -> bool

Perform LDLT Factorization (Lazy evaluation)

See also: *factorize()*

Template Parameters **Fn** –

Parameters **getA** – **[in]** function to access the elements of A

inline auto **is_spd()** const noexcept -> bool

Is A symmetric positive definite (spd)

Returns true

Returns false

auto **witness()** -> double

witness that certifies A is not symmetric positive definite (spd)

Returns auto

auto **sym_quad**(const Vec &A) const -> double

Calculate v'*{A}(p,p)*v.

Parameters **A** – **[in]**

Returns double

auto **sqrt()** -> Mat

Public Members

Rng **p** = {0U, 0U}

the rows where the process starts and stops

Vec **v**

witness vector

Class lmi0_oracle

- Defined in file_ellcpp_oracles_lmi0_oracle.hpp

Page Contents

Class Documentation

class **lmi0_oracle**

Oracle for Linear Matrix Inequality.

This oracle solves the following feasibility problem:

```
find  x
s.t.  F * x >= 0
```

Public Functions

inline explicit **lmi0_oracle**(gsl::span<const Arr> F)

Construct a new lmi0 oracle object.

Parameters **F** – **[in]**

auto **operator()**(const Arr &x) -> std::optional<Cut>

Parameters **x** – **[in]**

Returns std::optional<Cut>

Public Members

ldlt_ext **_Q**

Class lmi_old_oracle

- Defined in file_ellcpp_oracles_lmi_old_oracle.hpp

Page Contents

Class Documentation

class **lmi_old_oracle**

Oracle for Linear Matrix Inequality.

This oracle solves the following feasibility problem:

```
find  x
s.t.  (B - F * x) >= 0
```

Public Functions

inline **lmi_old_oracle**(gsl::span<const Arr> F, Arr B)

Construct a new lmi oracle object.

Parameters

- **F** – **[in]**
- **B** – **[in]**

auto **operator()**(const Arr &x) -> std::optional<Cut>

Parameters **x** – **[in]**

Returns std::optional<Cut>

Class lmi_oracle

- Defined in file_ellcpp_oracles_lmi_oracle.hpp

Page Contents

Class Documentation

class **lmi_oracle**

Oracle for Linear Matrix Inequality.

This oracle solves the following feasibility problem:

```
find  x
s.t.  (B - F * x) >= 0
```

Public Functions

inline **lmi_oracle**(gsl::span<const Arr> F, Arr B)

Construct a new lmi oracle object.

Parameters

- **F** – **[in]**
- **B** – **[in]**

auto **operator()**(const Arr &x) -> std::optional<Cut>

Parameters **x** – **[in]**

Returns std::optional<Cut>

Class lowpass_oracle

- Defined in file_ellcpp_oracles_lowpass_oracle.hpp

Page Contents

Class Documentation

class **lowpass_oracle**

Oracle for FIR lowpass filter design.

This example is taken from Almir Mutapcic in 2006:

```
min   \gamma
s.t.  L^2(\omega) \le R(\omega) \le U^2(\omega), \forall \omega \in
```

[0,] R() > 0, [0,]

Public Functions

inline **lowpass_oracle**(const Arr &Ap, const Arr &As, const Arr &Anr, double Lpsq, double Upsq)

Construct a new lowpass oracle object.

Parameters

- **Ap** – **[in]**
- **As** – **[in]**
- **Anr** – **[in]**
- **Lpsq** – **[in]**
- **Upsq** – **[in]**

auto **operator()**(const Arr &x, double &Spsq) const -> std::tuple<ParallelCut, bool>

Parameters

- **x** – **[in]**
- **Spsq** – **[in]**

Returns auto

Template Class monomial

- Defined in file_ellip_monomial.hpp

Page Contents

Template Parameter Order

1. `typename _Tp`

Class Documentation

template<typename **_Tp**>

class **monomial**

Reference: S.P. Boyd, S.-J. Kim and L.Vandenberghe and A. Hassibi. A Tutorial on Geometric Programming. Available at http://www.standford.edu/~boyd/gp_tutorial.html Monomial function object. Let x_1, . . . , x_n denote n real positive variable and x = (x_1, . . . , x_n) a vector with components x_i. A read valued function f of x, with the form

f(x0, . . . , xn-1) = c * x0^a1 * x2^a2 . . . xn-1^an-1

where c > 0 and $a_i R$, is called a monomial function, or more informally, a monomial (of the variables x_1, . . . , x_n). We refer to the constant c as the coefficient of the monmial, and we refer to the constants a_1, . . . a_n as the exponents of the monomial. As an example, $2.3 x_1^2 x_2^{-0.15}$ is a monomial of the variables x_1 and x_2, with coefficient 2.3 and x_2-exponent -0.15. Any positive constant is a monomial, as is any variable. Monomials are closed under multiplication and division: if f and g are both monomials then so are $f*g$ and f/g. (This includes scaling by any positive constant.) A monomial raise to any power is also a monomial.

The term `monomial', as used here (in the context of geometric programming) is similar to, but differs from the standard definition of `monomial' used in algebra. In algebra, a monomial has the form about, but the exponents a_i must be non-negative integers, and the coefficent c is one.

Todo:

: In current implementation, the exponent _a is represented by only <double>. The type should be _Tp in general such that it can be also be a <AAF>.

Public Functions

inline explicit **monomial**(size_t n)

Construct a monomial with *n* variables.

inline **monomial**(const *_Tp* &c, const Vec &a)

Constructor.

inline **monomial**(const *_Tp* &c, std::initializer_list<*_Tp*> lst)

Construct an array with an initializer_list of values.

inline **monomial**(size_t n, const *_Tp* &c)

Constructor

template<typename **_Up**, class **Map**>
monomial(const *monomial*<*_Up*> &mon, const *Map* &polarity)

Constructor (for AAF -> double)

inline **monomial**(size_t n, const *_Tp* ar[])

inline **~monomial**()

Destructor

inline _Self &**operator***=(const _Self &M)

Multiply and assign

inline _Self &**operator/**=(const _Self &M)

Divide and assign

inline _Self &**operator***=(const *_Tp* &c)

Multiply and assign

inline _Self &**operator/**=(const *_Tp* &c)

Divide and assign

inline _Self **operator***(const *_Tp* &c) const

Multiply

inline _Self **operator/**(const *_Tp* &c) const

Divide

inline void **sqrt**()

Square root

template<typename **_Up**>
inline *_Tp* **lse**(const std::valarray<*_Up*> &y) const

Function evaluation of log(f(exp(y))), i.e., res = b + dot(a,y).

template<typename **_Up**>
inline Vec **lse_gradient**(const *_Up*&) const

Gradient of log(f(exp(y)))

template<typename **_Up**>
inline *_Tp* **log_exp_fvalue_with_gradient**(const *_Up* &y, Vec &g) const

Function evaluation and gradient of log(f(exp(y))

template<typename **_Up**>
inline Vec **lse_gradient**(const *_Up* &y, *_Tp* &f) const

Function evaluation and gradient of log(f(exp(y))

inline void **set_coeff**(*_Tp* c)

template<>
monomial(const *monomial*<aaf> &mon, const pmap &polarity)

Constructor (for aaf -> double)

Public Members

_Tp **_b**

coefficient in log, i.e. log(c)

Vec **_a**

vector of exponents (a0, .. an-1)

Template Class negCycleFinder

- Defined in file_netoptim_neg_cycle.hpp

Page Contents

Template Parameter Order

1. `typename Graph`

Class Documentation

template<typename **Graph**>

class **negCycleFinder**

negative cycle

Negative cycle detection for weighed graphs.

a. BF needs a source node.
b. BF detect whether there is a negative cycle at the fianl stage.
c. BF restarts the solution (dist[u]) every time.

Template Parameters **Graph** – Note: Bellman-Ford's shortest-path algorithm (BF) is NOT the best way to detect negative cycles, because

Public Functions

inline explicit **negCycleFinder**(const *Graph* &G)

Construct a new neg Cycle Finder object.

Parameters **G** – **[in]**

template<typename **Container**, typename **WeightFn**>
inline auto **find_neg_cycle**(*Container* &&dist, *WeightFn* &&get_weight) -> std::vector<edge_t>

find negative cycle

Template Parameters

- **Container** –
- **WeightFn** –

Parameters

- **dist** – **[inout]**
- **get_weight** – **[in]**

Returns std::vector<edge_t>

Template Class network_oracle

- Defined in file_ellcpp_oracles_network_oracle.hpp

Page Contents

Template Parameter Order

1. `typename Graph`
2. `typename Container`
3. `typename Fn`

Class Documentation

template<typename **Graph**, typename **Container**, typename **Fn**>

class **network_oracle**

Oracle for Parametric Network Problem.

This oracle solves the following feasibility problem:

```
find    x, u
s.t.    u[j] - u[i] \le h(e, x)
        \forall e(i, j) \in E
```

Template Parameters

- **Graph** –
- **Container** –
- **Fn** –

Public Functions

inline **network_oracle**(const *Graph* &G, *Container* &u, *Fn* h)

Construct a new network oracle object.

Parameters

- **G** – **[in]** a directed graph (V, E)
- **u** – **[inout]** list or dictionary
- **h** – **[in]** function evaluation and gradient

explicit **network_oracle**(const *network_oracle*&) = default

Construct a new network oracle object.

template<typename **opt_type**>
inline auto **update**(const *opt_type* &t) -> void

Parameters **t** – **[in]** the best-so-far optimal value

template<typename **T**>
inline auto **operator()**(const *T* &x) -> std::optional<std::tuple<*T*, double>>

Make object callable for cutting_plane_feas()

Template Parameters **T** –

Parameters **x** – **[in]**

Returns std::optional<std::tuple<T, double>>

Template Class optscaling_oracle

- Defined in file_ellcpp_oracles_optscaling_oracle.hpp

Page Contents

Nested Relationships

Nested Types

- *Class optscaling_oracle::Ratio*

Template Parameter Order

1. `typename Graph`
2. `typename Container`
3. `typename Fn`

Class Documentation

template<typename **Graph**, typename **Container**, typename **Fn**>

class **optscaling_oracle**

Oracle for Optimal Matrix Scaling.

This example is taken from [Orlin and Rothblum, 1985]:

```
min     \pi/\phi
s.t.    \phi \le u[i] * |aij| * u[j]^-1 \le \pi,
        \forall aij != 0,
        \pi, \phi, u, positive
```

Template Parameters

- **Graph** –
- **Container** –
- **Fn** –

Public Functions

inline **optscaling_oracle**(const *Graph* &G, *Container* &u, *Fn* get_cost)

Construct a new optscaling oracle object.

Parameters

- **G** – **[in]**
- **u** – **[inout]**
- **get_cost** – **[in]**

explicit **optscaling_oracle**(const *optscaling_oracle*&) = default

Construct a new optscaling oracle object.

inline auto **operator()**(const Arr &x, double &t) -> std::tuple<Cut, bool>

Make object callable for cutting_plane_dc()

See also:

cutting_plane_dc

Parameters

- **x** – **[in]** (,) in log scale
- **t** – **[in]** the best-so-far optimal value

Returns std::tuple<Cut, double>

Class optscaling_oracle::Ratio

- Defined in file_ellcpp_oracles_optscaling_oracle.hpp

Page Contents

Nested Relationships

This class is a nested type of *Template Class optscaling_oracle*.

Class Documentation

class **Ratio**

Ratio.

Public Functions

inline **Ratio**(const Graph &G, Fn get_cost)

Construct a new Ratio object.

Parameters

- **G** – **[in]**
- **get_cost** – **[in]**

explicit **Ratio**(const *Ratio*&) = default

Construct a new Ratio object (only explicitly)

inline auto **eval**(const edge_t &e, const Arr &x) const -> double

Evaluate function.

Parameters

- **e** – **[in]**
- **x** – **[in]** (,) in log scale

Returns double

inline auto **grad**(const edge_t &e, const Arr &x) const -> Arr

Gradient function.

Parameters

- **e** – **[in]**
- **x** – **[in]** (,) in log scale

Returns Arr

Template Class posynomial

- Defined in file_ellip_posynomial.hpp

Page Contents

Template Parameter Order

1. `typename _Tp`

Class Documentation

template<typename **_Tp**>

class **posynomial**

Reference: S.P. Boyd, S.J. Kim and L.Vandenberghe and A. Hassibi. A Tutorial on Geometric Programming. Available at http://www.standford.edu/~boyd/gp_tutorial.html Posynomial function. A sum of one or more monomials, i.e., a function of the form

f(x) = {k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} … x_n^{a_{nk}},

where c_k > 0, is called a posynomial function or, more simply, a posynomial (with K terms, in the variables x_1, …, x_n. The term `posynomial' is meant to suggest a combination of `positive' and `polynomial'.

Any monomial is also a posynomial. Posynomials are close under addition, multiplication, and positive scaling. Posynomials can be divided by monomials (with the result also a posynomial): If f is a posynomial and g is a monomial, then f/g is a posynomial. Note that <_Tp> could be <double> or <AAF>

Public Functions

inline explicit **posynomial**(size_t n, size_t N)

Constructor

inline **posynomial**(const *monomial*<*_Tp*> &m)

Constructor

template<typename **Up**, class **Map**>
inline **posynomial**(const *posynomial*<*Up*> &posyn, const *Map* &polarity)

Constructor (for AAF -> double)

inline **~posynomial**()

Destructor

inline _Self &**operator+=**(const *monomial*<*_Tp*> &m)

Add and assign

inline _Self &**operator+=**(const _Self &P)

Add and assign

inline _Self &**operator*=**(const *monomial*<*_Tp*> &m)

Multiply and assign

inline _Self &**operator/=**(const *monomial*<*_Tp*> &m)

Divide and assign

inline _Self &**operator*=**(const *_Tp* &c)

Multiply and assign

inline _Self &**operator/=**(const *_Tp* &c)

Divide and assign

inline _Self **operator***(const _Self &P) const

Multiply.

Todo:

simplify the result.

template<typename **_Up**>
inline *_Tp* **lse**(const *_Up* &y) const

Todo:

should combine the following two functions into one, and eniminate _p

Function evaluation of log(f(exp(y))).

template<typename **_Up**>
inline std::valarray<*_Tp*> **log_exp_gradient**(const *_Up* &y) const

Gradient of log(f(exp(y))). Precondition: call f(y) previously

template<typename **_Up**>
inline *_Tp* **log_exp_fvalue_with_gradient**(const *_Up* &y, std::valarray<*_Tp*> &g) const

function value and gradient of log(f(exp(y))). Note that for AAF, the two quantities have to be evaluated *at the same time* in order to maintain the noise symbols consistency

template<typename **_Up**>
inline std::valarray<*_Tp*> **lse_gradient**(const *_Up* &y, *_Tp* &f) const

function value and gradient of log(f(exp(y))).

inline **posynomial**(const _Self &Q)

inline _Self &**operator=**(const _Self &Q)

template<>
posynomial(const *posynomial*<aaf> &posyn, const pmap &polarity)

Constructor (for AAF -> double)

Public Members

std::vector<*monomial*<*_Tp*>> **_M**

vector of monomials

Template Class profit_max

- Defined in file_ellip_profitmaxprob.hpp

Page Contents

Inheritance Relationships

Base Type

- `public gp_base< _Tp >` (*Template Class gp_base*)

Template Parameter Order

1. `typename _Tp`

Class Documentation

template<typename **_Tp**>

class **profit_max** : public *gp_base*<*_Tp*>

Profit Maximization Problem, taken from:

[1] Aliabadi, Hossein, and Maziar Salahi. "Robust Geometric Programming

Approach to Profit Maximization with Interval Uncertainty." Computer Science Journal of Moldova 21.1 (2013): 61.

Public Functions

profit_max()

~profit_max()

inline void **gp_setup**()

Problem formulation.

inline *_Tp* **obj**(const std::valarray<double> &z)

objective function we want to analyse

template<>

profit_max()

Profit Maximization Problem, taken from:

[1] Aliabadi, Hossein, and Maziar Salahi. "Robust Geometric Programming

Approach to Profit Maximization with Interval Uncertainty." Computer Science Journal of Moldova 21.1 (2013): 61.

template<>
~profit_max()

template<>
profit_max()

Profit Maximization Problem, taken from:

[1] Aliabadi, Hossein, and Maziar Salahi. "Robust Geometric Programming

Approach to Profit Maximization with Interval Uncertainty." Computer Science Journal of Moldova 21.1 (2013): 61.

template<>
~profit_max()

Class profit_oracle

- Defined in file_ellcpp_oracles_profit_oracle.hpp

Page Contents

Class Documentation

class **profit_oracle**

Oracle for a profit maximization problem.

This example is taken from [Aliabadi and Salahi, 2013]:

```
max     p(A x1^alpha x2^beta) - v1*x1 - v2*x2
s.t.    x1 \le k
```

where:

```
p(A x1^alpha x2^beta): Cobb-Douglas production function
p: the market price per unit
A: the scale of production
alpha, beta: the output elasticities
x: input quantity
v: output price
k: a given constant that restricts the quantity of x1
```

Public Functions

inline **profit_oracle**(double p, double A, double k, const Arr &a, const Arr &v)

Construct a new profit oracle object.

Parameters

- **p** – **[in]** the market price per unit
- **A** – **[in]** the scale of production
- **k** – **[in]** a given constant that restricts the quantity of x1
- **a** – **[in]** the output elasticities
- **v** – **[in]** output price

explicit **profit_oracle**(const *profit_oracle*&) = default

Construct a new profit oracle object (only explicitly)

auto **operator()**(const Arr &y, double &t) const -> std::tuple<Cut, bool>

Parameters

- **y** – **[in]** input quantity (in log scale)
- **t** – **[in]** the best-so-far optimal value

Returns std::tuple<Cut, double> Cut and the updated best-so-far value

Public Members

Arr **_a**

Class profit_q_oracle

- Defined in file_ellcpp_oracles_profit_oracle.hpp

Page Contents

Class Documentation

class **profit_q_oracle**

Oracle for profit maximization problem (discrete version)

This example is taken from [Aliabadi and Salahi, 2013]

```
max     p(A x1^alpha x2^beta) - v1*x1 - v2*x2
s.t.    x1 \le k
```

where:

```
p(A x1^alpha x2^beta): Cobb-Douglas production function
p: the market price per unit
A: the scale of production
alpha, beta: the output elasticities
x: input quantity (must be integer value)
v: output price
k: a given constant that restricts the quantity of x1
```

See also:

profit_oracle

Public Functions

inline **profit_q_oracle**(double p, double A, double k, const Arr &a, const Arr &v)

Construct a new profit q oracle object.

Parameters

- **p** – **[in]** the market price per unit
- **A** – **[in]** the scale of production
- **k** – **[in]** a given constant that restricts the quantity of x1
- **a** – **[in]** the output elasticities
- **v** – **[in]** output price

auto **operator()**(const Arr &y, double &t, bool retry) -> std::tuple<Cut, Arr, bool, bool>

Make object callable for cutting_plane_q()

See also:

cutting_plane_q

Parameters

- **y** – **[in]** input quantity (in log scale)
- **t** – **[in]** the best-so-far optimal value

Returns Cut and the updated best-so-far value

Class profit_rb_oracle

- Defined in file_ellcpp_oracles_profit_oracle.hpp

Page Contents

Class Documentation

class **profit_rb_oracle**

Oracle for a profit maximization problem (robust version)

This example is taken from [Aliabadi and Salahi, 2013]:

```
max     p'(A x1^alpha' x2^beta') - v1'*x1 - v2'*x2
s.t.    x1 \le k'
```

where: alpha' = alpha e1 beta' = beta e2 p' = p e3 k' = k e4 v' = v e5

See also:

profit_oracle

Public Functions

inline **profit_rb_oracle**(double p, double A, double k, const Arr &a, const Arr &v, const Arr &e, double e3)

Construct a new profit rb oracle object.

Parameters

- **p** – **[in]** the market price per unit
- **A** – **[in]** the scale of production
- **k** – **[in]** a given constant that restricts the quantity of x1
- **a** – **[in]** the output elasticities
- **v** – **[in]** output price
- **e** – **[in]** paramters for uncertainty
- **e3** – **[in]** paramters for uncertainty

inline auto **operator()**(const Arr &y, double &t)

Make object callable for cutting_plane_dc()

See also:

cutting_plane_dc

Parameters

- **y** – **[in]** input quantity (in log scale)
- **t** – **[in]** the best-so-far optimal value

Returns Cut and the updated best-so-far value

Template Class dict

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- `public std::unordered_map< Key, T >`

Template Parameter Order

1. `typename Key`
2. `typename T`

Class Documentation

template<typename **Key**, typename **T**>

class **dict** : public std::unordered_map<*Key*, *T*>

Template Parameters

- **Key** –
- **T** –

Public Types

using **value_type** = std::pair<const *Key*, *T*>

Public Functions

inline **dict**()

Construct a new dict object.

auto **operator=**(Self&&) noexcept -> *dict*& = default

Returns _Self&

dict(*dict*<*Key*, *T*>&&) noexcept = default

Move Constructor (default)

~dict() = default

dict(const *dict*<*Key*, *T*>&) = default

Construct a new dict object.

Copy through explicitly the public copy() function!!!

Template Class set

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- `public std::unordered_set< Key >`

Template Parameter Order

1. `typename Key`

Class Documentation

template<typename **Key**>

class **set** : public std::unordered_set<*Key*>

Template Parameters **Key** –

Public Functions

inline **set**()

Construct a new set object.

auto **operator=**(*set*&&) noexcept -> *set*& = default

Returns _Self&

set(*set*<*Key*>&&) noexcept = default

Move Constructor (default)

set(const *set*<*Key*>&) = default

Copy Constructor (deleted)

Copy through explicitly the public copy() function!!!

Class qmi_oracle

- Defined in file_ellcpp_oracles_qmi_oracle.hpp

Page Contents

Class Documentation

class **qmi_oracle**

Oracle for Quadratic Matrix Inequality.

This oracle solves the following feasibility problem:

```
find  x
s.t.  t * I - F(x)' F(x) >= 0
```

where

```
F(x) = F0 - (F1 * x1 + F2 * x2 + ...)
```

Public Functions

inline **qmi_oracle**(gsl::span<const Arr> F, Arr F0)

Construct a new qmi oracle object.

Parameters

- **F** – **[in]**
- **F0** – **[in]**

inline auto **update**(double t) -> void

Update t.

Parameters **t** – **[in]** the best-so-far optimal value

auto **operator()**(const Arr &x) -> std::optional<Cut>

Parameters **x** – **[in]**

Returns std::optional<Cut>

Public Members

ldlt_ext **_Q**

Template Class AtlasView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Template Parameter Order

1. `typename Vertex`
2. `typename Graph`

Class Documentation

template<typename **Vertex**, typename **Graph**>

class **AtlasView**

Template Parameters

- **Vertex** –
- **Graph** –

Public Functions

inline **AtlasView**(*Vertex* v, const *Graph* &G)

Construct a new Atlas View object.

Parameters

- **v** – **[in]**
- **G** – **[in]**

inline auto **begin**() const

Returns auto

inline auto **end**() const

Returns auto

inline auto **cbegin**() const

Returns auto

inline auto **cend**() const

Returns auto

Template Class DiGraphS

- Defined in file_xnetwork_classes_digraphs.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::Graph< nodeview_t, adjlist_t >` (*Template Class Graph*)

Template Parameter Order

1. `typename nodeview_t`
2. `typename adjlist_t`

Class Documentation

template<typename **nodeview_t**, typename **adjlist_t** = py::*dict*<*Value_type*<*nodeview_t*>, int>>

class **DiGraphS** : public xn::*Graph*<*nodeview_t*, *adjlist_t*>

Base class for directed graphs.

A *DiGraphS* stores nodes and edges with optional data, or attributes.

DiGraphSs hold directed edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) C++ objects with optional key/value attributes. By convention `None` is not used as a node.

Edges are represented as links between nodes with optional key/value attributes.

Parameters

node_container : input graph (optional, default: None) Data to initialize graph. If None (default) an empty graph is created. The data can be any format that is supported by the to_networkx_graph() function, currently including edge list, dict of dicts, dict of lists, NetworkX graph, NumPy matrix or 2d ndarray, SciPy sparse matrix, or PyGraphviz graph.

See Also

Graph DiGraph MultiGraph MultiDiGraph OrderedGraph

Examples

Create an empty graph structure (a "null graph") with 5 nodes and no edges.

auto v = std::vector{3, 4, 2, 8}; auto G = *xn::DiGraphS(v)*;

auto va = *py::dict*{{3, 0.1}, {4, 0.5}, {2, 0.2}}; auto G = *xn::DiGraphS(va)*;

auto r = py::range(100); auto G = *xn::DiGraphS(r)*;

G can be grown in several ways.

Nodes:**

Add one node at a time:

G.add_node(1)

Add the nodes from any container (a list, dict, set or even the lines from a file or the nodes from another graph).

G.add_nodes_from([2, 3]) G.add_nodes_from(range(100, 110)) H = xn::path_graph(10) G.add_nodes_from(H)

In addition to strings and integers any hashable C++ object (except None) can represent a node, e.g. a customized node object, or even another *DiGraphS*.

G.add_node(H)

Edges:**

G can also be grown by adding edges.

Add one edge,

G.add_edge(1, 2);

a list of edges,

G.add_edges_from([(1, 2), (1, 3)]);

or a collection of edges,

G.add_edges_from(H.edges());

If some edges connect nodes not yet in the graph, the nodes are added automatically. There are no errors when adding nodes or edges that already exist.

Attributes:**

Each graph can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using direct manipulation of the attribute dictionaries named graph, node and edge respectively.

G.graph["day"] = boost::any("Friday");

{'day': 'Friday'}

Subclasses (Advanced):**

The *DiGraphS* class uses a container-of-container-of-container data structure. The outer dict (node_dict) holds adjacency information keyed by node. The next dict (adjlist_dict) represents the adjacency information and holds edge data keyed by neighbor. The inner dict (edge_attr_dict) represents the edge data and holds edge attribute values keyed by attribute names.

Each of these three dicts can be replaced in a subclass by a user defined dict-like object. In general, the dict-like features should be maintained but extra features can be added. To replace one of the dicts create a new graph class by changing the class(!) variable holding the factory for that dict-like structure. The variable names are node_dict_factory, node_attr_dict_factory, adjlist_inner_dict_factory, adjlist_outer_dict_factory, edge_attr_dict_factory and graph_attr_dict_factory.

node_dict_factory : function, (default: dict) Factory function to be used to create the dict containing node attributes, keyed by node id. It should require no arguments and return a dict-like object

node_attr_dict_factory: function, (default: dict) Factory function to be used to create the node attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object

adjlist_outer_dict_factory : function, (default: dict) Factory function to be used to create the outer-most dict in the data structure that holds adjacency info keyed by node. It should require no arguments and return a dict-like object.

adjlist_inner_dict_factory : function, (default: dict) Factory function to be used to create the adjacency list dict which holds edge data keyed by neighbor. It should require no arguments and return a dict-like object

edge_attr_dict_factory : function, (default: dict) Factory function to be used to create the edge attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

graph_attr_dict_factory : function, (default: dict) Factory function to be used to create the graph attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

Typically, if your extension doesn't impact the data structure all methods will inherit without issue except: `to_directed/to_undirected`. By default these methods create a DiGraph/DiGraphS class and you probably want them to create your extension of a DiGraph/DiGraphS. To facilitate this we define two class variables that you can set in your subclass.

to_directed_class : callable, (default: DiGraph or MultiDiGraph) Class to create a new graph structure in the `to_directed` method. If `None`, a NetworkX class (DiGraph or MultiDiGraph) is used.

to_undirected_class : callable, (default: *DiGraphS* or MultiGraph) Class to create a new graph structure in the `to_undirected` method. If `None`, a NetworkX class (*DiGraphS* or MultiGraph) is used.

Examples

Create a low memory graph class that effectively disallows edge attributes by using a single attribute dict for all edges. This reduces the memory used, but you lose edge attributes.

class ThinGraph(xn::DiGraphS):

... all_edge_dict = {'weight': 1} ... def single_edge_dict(self): ... return self.all_edge_dict ... edge_attr_dict_factory = single_edge_dict

G = ThinGraph() G.add_edge(2, 1) G[2][1]

{'weight': 1}

G.add_edge(2, 2) G[2][1] is G[2][2]

True

Please see :mod:~`networkx.classes.ordered` for more examples of creating graph subclasses by overwriting the base class `dict` with a dictionary-like object.

Public Types

using **`Node`** = typename _Base::Node

using **`edge_t`** = std::pair<*Node*, *Node*>

using **`graph_attr_dict_factory`** = typename _Base::graph_attr_dict_factory

using **`adjlist_outer_dict_factory`** = typename _Base::adjlist_outer_dict_factory

using **`key_type`** = typename _Base::key_type

using **`value_type`** = typename _Base::value_type

using **`coro_t`** = boost::coroutines2::coroutine<*edge_t*>

using **`pull_t`** = typename *coro_t*::pull_type

Public Functions

inline explicit **DiGraphS**(const *nodeview_t* &Nodes)

Initialize a graph with edges, name, or graph attributes.

Parameters

node_container : input nodes

Examples

v = std::vector{5, 3, 2}; G = *xn::DiGraphS(v)*; // or DiGraph, MultiGraph, MultiDiGraph, etc

r = py::range(100); G = *xn::DiGraphS(r, r)*; // or DiGraph, MultiGraph, MultiDiGraph,

etc

inline explicit **DiGraphS**(int num_nodes)

inline auto **adj**() const

DiGraphS adjacency object holding the neighbors of each node.

This object is a read-only dict-like structure with node keys and neighbor-dict values. The neighbor-dict is keyed by neighbor to the edge-data-dict. So `G.adj[3][2]['color'] = 'blue'`sets the color of the edge(3, 2)to"blue"`.

Iterating over G.adj behaves like a dict. Useful idioms include `for nbr, datadict in G.adj[n].items():`.

The neighbor information is also provided by subscripting the graph. So `for nbr, foovalue in G[node].data('foo', default=1):` works.

For directed graphs, `G.adj` holds outgoing (successor) info.

inline auto **succ**() const

Graph adjacency object holding the successors of each node.

This object is a read-only dict-like structure with node keys and neighbor-dict values. The neighbor-dict is keyed by neighbor to the edge-data-dict. So `G.succ[3][2]['color'] = 'blue'`sets the color of the edge(3, 2)to"blue"`.

Iterating over G.succ behaves like a dict. Useful idioms include `for nbr, datadict in G.succ[n].items():`. A data-view not provided by dicts also exists: `for nbr, foovalue in G.succ[node].data('foo'): and a default can be set via adefaultargument to thedata` method.

The neighbor information is also provided by subscripting the graph. So `for nbr, foovalue in G[node].data('foo', default=1):` works.

For directed graphs, `G.adj` is identical to `G.succ`.

template<typename **U** = *key_type*>

inline std::enable_if<std::is_same<*U*, *value_type*>::value>::type **add_edge**(const *Node* &u, const *Node* &v)

Add an edge between u and v.

The nodes u and v will be automatically added if (they are not already : the graph.

Edge attributes can be specified with keywords or by directly accessing the edge"s attribute dictionary. See examples below.

Parameters

u, v : nodes Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) C++ objects.

See Also

add_edges_from : add a collection of edges

Notes

Adding an edge that already exists updates the edge data.

Many XNetwork algorithms designed for weighted graphs use an edge attribute (by default `weight`) to hold a numerical value.

Examples

The following all add the edge e=(1, 2) to graph G) {

G = *xn::DiGraphS()* // or DiGraph, MultiGraph, MultiDiGraph, etc e = (1, 2); G.add_edge(1, 2) // explicit two-node form G.add_edges_from([(1, 2)]); // add edges from iterable container

Associate data to edges using keywords) {

G.add_edge(1, 2);

For non-string attribute keys, use subscript notation.

G.add_edge(1, 2); G[1][2].update({0: 5}); G.edges()[1, 2].update({0: 5});

template<typename **U** = *key_type*>
inline std::enable_if<!std::is_same<*U*, *value_type*>::value>::type **add_edge**(const *Node* &u, const *Node* &v)

template<typename **T**>
inline auto **add_edge**(const *Node* &u, const *Node* &v, const *T* &data)

template<typename **C1**, typename **C2**>
inline auto **add_edges_from**(const *C1* &edges, const *C2* &data)

inline auto **has_successor**(const *Node* &u, const *Node* &v) -> bool

Returns True if node u has successor v.

This is true if graph has the edge u->v.

inline auto &**successors**(const *Node* &n)

Returns an iterator over successor nodes of n.

A successor of n is a node m such that there exists a directed edge from n to m.

Parameters

n : node A node in the graph

Raises

NetworkXError If n is not in the graph.

See Also

predecessors

Notes

neighbors() and *successors()* are the same.

inline const auto &**successors**(const *Node* &n) const

inline auto **edges**() const -> *pull_t*

An OutEdgeView of the DiGraph as G.edges().

edges(self, nbunch=None, data=False, default=None)

The OutEdgeView provides set-like operations on the edge-tuples as well as edge attribute lookup. When called, it also provides an EdgeDataView object which allows control of access to edge attributes (but does not provide set-like operations). Hence, `G.*edges()*[u, v]['color']`provides the value of the color attribute for edge`(u, v)`while` for (u, v, c) in G.edges().data('color', default='red'):` iterates through all the edges yielding the color attribute with default `'red'` if no color attribute exists.

Parameters

nbunch : single node, container, or all nodes (default= all nodes) The view will only report edges incident to these nodes. data : string or bool, optional (default=False) The edge attribute returned in 3-tuple (u, v, ddict[data]). If True, return edge attribute dict in 3-tuple (u, v, ddict). If False, return 2-tuple (u, v). default : value, optional (default=None) Value used for edges that don't have the requested attribute. Only relevant if data is not True or False.

Returns

edges : OutEdgeView A view of edge attributes, usually it iterates over (u, v) or (u, v, d) tuples of edges, but can also be used for attribute lookup as `edges[u, v]['foo']`.

See Also

in_edges, out_edges

Notes

Nodes in nbunch that are not in the graph will be (quietly) ignored. For directed graphs this returns the out-edges.

Examples

G = nx.DiGraph() # or MultiDiGraph, etc nx.add_path(G, [0, 1, 2]) G.add_edge(2, 3, weight=5) [e for e in G.edges()]

[(0, 1), (1, 2), (2, 3)]

G.edges().data() # default data is {} (empty dict)

OutEdgeDataView([(0, 1, {}), (1, 2, {}), (2, 3, {'weight': 5})])

G.edges().data('weight', default=1)

OutEdgeDataView([(0, 1, 1), (1, 2, 1), (2, 3, 5)])

G.edges()([0, 2]) # only edges incident to these nodes

OutEdgeDataView([(0, 1), (2, 3)])

G.edges()(0) # only edges incident to a single node (use G.adj[0]?)

OutEdgeDataView([(0, 1)])

inline auto **degree**(const *Node* &n) const

inline auto **clear**()

Remove all nodes and edges from the graph.

This also removes the name, and all graph, node, and edge attributes.

Examples

G = xn::path_graph(4); // or DiGraph, MultiGraph, MultiDiGraph, etc G.clear(); list(G.nodes);

[];

list(G.edges());

[];

inline auto **is_multigraph**()

Return true if (graph is a multigraph, false otherwise.

inline auto **is_directed**()

Return true if (graph is directed, false otherwise.

Public Members

adjlist_outer_dict_factory &**_succ**

Template Class EdgeView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Template Parameter Order

1. `typename Graph`

Class Documentation

template<typename **Graph**>

class **EdgeView**

Template Parameters **Graph** –

Public Functions

inline explicit **EdgeView**(const *Graph* &G)

Construct a new Edge View object.

Parameters **G** – **[in]**

inline auto **begin**() const

Returns auto

inline auto **end**() const

Returns auto

inline auto **cbegin**() const

Returns auto

inline auto **cend**() const

Returns auto

Template Class grAdaptor

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::VertexView< Graph >` (*Template Class VertexView*)

Template Parameter Order

1. **`Graph <exhale_class_classxn_1_1Graph_>`_**

Class Documentation

template<typename **Graph**>

class **grAdaptor** : public xn::*VertexView*<*Graph*>

Template Parameters **Graph** –

Public Types

using **Vertex** = typename boost::graph_traits<*Graph*>::vertex_descriptor

using **node_t** = *Vertex*

using **edge_t** = typename boost::graph_traits<*Graph*>::edge_descriptor

Public Functions

grAdaptor() = delete

Construct a new gr Adaptor object.

inline explicit **grAdaptor**(*Graph* &&G) noexcept

Construct a new gr Adaptor object.

Parameters G – **[in]**

Template Class Graph

- Defined in file_xnetwork_classes_graph.hpp

Page Contents

Inheritance Relationships

Base Type

- public xn::object (*Struct object*)

Derived Type

- public xn::VertexView< Graph > (*Template Class VertexView*)

Template Parameter Order

1. typename __nodeview_t
2. typename adjlist_t

Class Documentation

template<typename **__nodeview_t**, typename **adjlist_t** = py::*set*<*Value_type*<*__nodeview_t*>>>

class **Graph** : public xn::*object*

Subclassed by *xn::VertexView< Graph >*

Public Types

using **nodeview_t** = *__nodeview_t*

using **Node** = typename *nodeview_t*::value_type

using **dict** = py::*dict*<const char*, boost::any>

using **graph_attr_dict_factory** = *dict*

using **adjlist_inner_dict_factory** = *adjlist_t*

using **adjlist_outer_dict_factory** = std::vector<*adjlist_t*>

using **key_type** = typename *adjlist_t*::key_type

using **value_type** = typename *adjlist_t*::value_type

using **edge_t** = std::pair<*Node*, *Node*>

using **node_t** = *Node*

Public Functions

inline explicit **Graph**(const *nodeview_t* &Nodes)

Initialize a graph with edges, name, or graph attributes.

Parameters

node_container : input nodes

Examples

v = std::vector{5, 3, 2}; G = *xn::Graph(v)*; // or DiGraph, MultiGraph, MultiDiGraph, etc

r = py::range(100); G = *xn::Graph(r)*; // or DiGraph, MultiGraph, MultiDiGraph, etc

inline explicit **Graph**(int num_nodes)

Graph(const *Graph*&) = delete

Graph &**operator=**(const *Graph*&) = delete

Graph(*Graph*&&) noexcept = default

inline auto **adj**() const

Graph adjacency object holding the neighbors of each node.

This object is a read-only dict-like structure with node keys and neighbor-dict values. The neighbor-dict is keyed by neighbor to the edge-data-dict. So `G.adj[3][2]['color'] = 'blue'`sets the color of the edge(3, 2)to"blue"`.

Iterating over G.adj behaves like a dict. Useful idioms include `for nbr, datadict in G.adj[n].items():`.

The neighbor information is also provided by subscripting the graph. So `for nbr, foovalue in G[node].data('foo', default=1):` works.

For directed graphs, `G.adj` holds outgoing (successor) info.

inline auto **adj**()

inline auto **_nodes_nbrs**() const

inline *Node* **null_vertex**() const

inline auto **get_name**()

inline auto **set_name**(boost::string_view s)

inline auto **begin**() const

Iterate over the nodes. Use: "for (auto&& n : G)".

Returns

niter : iterator An iterator over all nodes : the graph.

Examples

G = xn::path_graph(4); // or DiGraph, MultiGraph, MultiDiGraph, etc [n for n : G];

[0, 1, 2, 3];

list(G);

[0, 1, 2, 3];

inline auto **end**() const

inline bool **contains**(const *Node* &n)

Return true if (n is a node, false otherwise. Use: "n : G".

Examples

G = xn::path_graph(4); // or DiGraph, MultiGraph, MultiDiGraph, etc 1 : G

true

inline const auto &**operator[]**(const *Node* &n) const

Return a dict of neighbors of node n. Use: "G[n]".

Parameters

n : node A node in the graph.

Returns

adj_dict : dictionary The adjacency dictionary for nodes connected to n.

Notes

G[n] is the same as G.adj[n] and similar to G.neighbors(n); (which is an iterator over G.adj[n]);

Examples

G = xn::path_graph(4); // or DiGraph, MultiGraph, MultiDiGraph, etc G[0];

AtlasView({1: {}});

inline auto &**operator[]**(const *Node* &n)

inline auto **nodes**()

inline auto **number_of_nodes**() const

Return the number of nodes : the graph.

Returns

nnodes : int The number of nodes : the graph.

See Also

order, **len** which are identical

Examples

G = xn::path_graph(3); // or DiGraph, MultiGraph, MultiDiGraph, etc len(G);

3

inline auto **number_of_edges**() const

inline auto **order**()

Return the number of nodes : the graph.

Returns

nnodes : int The number of nodes : the graph.

See Also

number_of_nodes, **len** which are identical

inline auto **has_node**(const *Node* &n)

Return true if (the graph contains the node n.

Identical to `n : G`

Parameters

n : node

Examples

```
G = xn::path_graph(3); // or DiGraph, MultiGraph, MultiDiGraph, etc
G.has_node(0);
```

true

template<typename **U** = *key_type*>
inline std::enable_if<std::is_same<*U*, *value_type*>::value>::type **add_edge**(const *Node* &u, const *Node* &v)

Add an edge between u and v.

The nodes u and v will be automatically added if (they are not already : the graph.

Edge attributes can be specified with keywords or by directly accessing the edge"s attribute dictionary. See examples below.

Parameters

u, v : nodes Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) C++ objects.

See Also

add_edges_from : add a collection of edges

Notes

Adding an edge that already exists updates the edge data.

Many XNetwork algorithms designed for weighted graphs use an edge attribute (by default `weight`) to hold a numerical value.

Examples

The following all add the edge e=(1, 2) to graph G) {

G = *xn::Graph()* // or DiGraph, MultiGraph, MultiDiGraph, etc e = (1, 2); G.add_edge(1, 2) // explicit two-node form G.add_edges_from([(1, 2)]); // add edges from iterable container

Associate data to edges using keywords) {

G.add_edge(1, 2);

For non-string attribute keys, use subscript notation.

G.add_edge(1, 2); G[1][2].update({0: 5}); G.edges()[1, 2].update({0: 5});

template<typename **U** = *key_type*>
inline std::enable_if<!std::is_same<*U*, *value_type*>::value>::type **add_edge**(const *Node* &u, const *Node* &v)

template<typename **T**>
inline auto **add_edge**(const *Node* &u, const *Node* &v, const *T* &data)

template<typename **C1**, typename **C2**>
inline auto **add_edges_from**(const *C1* &edges, const *C2* &data)

inline auto **has_edge**(const *Node* &u, const *Node* &v) -> bool

inline auto **degree**(const *Node* &n) const

inline auto **clear**()

An *EdgeView* of the *Graph* as G.edges().

edges(nbunch=None, data=false, default=None);

The *EdgeView* provides set-like operations on the edge-tuples as well as edge attribute lookup. When called, it also provides an EdgeDataView object which allows control of access to edge attributes (but does not provide set-like operations). Hence, `G.edges[u, v]["color"]` provides the value of the color attribute for edge `(u, v)` while `for (auto [u, v, c] : G.edges.data("color", default="red") {` iterates through all the edges yielding the color attribute with default `"red"` if (no color attribute exists.

Parameters

nbunch : single node, container, or all nodes (default= all nodes); The view will only report edges incident to these nodes. data : string or bool, optional (default=false); The edge attribute returned : 3-tuple (u, v, ddict[data]). If true, return edge attribute dict : 3-tuple (u, v, ddict). If false, return 2-tuple (u, v). default : value, optional (default=None); Value used for edges that don''t have the requested attribute. Only relevant if (data is not true or false.

Returns

edges : *EdgeView* A view of edge attributes, usually it iterates over (u, v); or (u, v, d) tuples of edges, but can also be used for attribute lookup as `edges[u, v]["foo"]`.

Notes

Nodes : nbunch that are not : the graph will be (quietly) ignored. For directed graphs this returns the out-edges.

Examples

G = xn::path_graph(3) // or MultiGraph, etc G.add_edge(2, 3, weight=5); [e for e : G.edges];

[(0, 1), (1, 2), (2, 3)];

G.edges.data(); // default data is {} (empty dict);

EdgeDataView([(0, 1, {}), (1, 2, {}), (2, 3, {"weight": 5})]);

G.edges.data("weight", default=1);

EdgeDataView([(0, 1, 1), (1, 2, 1), (2, 3, 5)]);

G.edges([0, 3]); // only edges incident to these nodes

EdgeDataView([(0, 1), (3, 2)]);

G.edges(0); // only edges incident to a single node (use

G.adj[0]?); EdgeDataView([(0, 1)]);

```
A DegreeView for the Graph as G.degree or G.degree().
```

The node degree is the number of edges adjacent to the node. The weighted node degree is the sum of the edge weights for edges incident to that node.

This object provides an iterator for (node, degree) as well as lookup for the degree for a single node.

Parameters

nbunch : single node, container, or all nodes (default= all nodes); The view will only report edges incident to these nodes.

weight : string or None, optional (default=None); The name of an edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.

Returns

If a single node is requested deg : int Degree of the node

OR if (multiple nodes are requested nd_view : A DegreeView object capable of iterating (node, degree) pairs

Examples

```
G = xn::path_graph(4); // or DiGraph, MultiGraph, MultiDiGraph,
```

etc

```
G.degree[0]; // node 0 has degree 1
```

1

```
list(G.degree([0, 1, 2]));
```

[(0, 1), (1, 2), (2, 2)];

inline auto **is_multigraph()**

Return true if (graph is a multigraph, false otherwise.

inline auto **is_directed()**

Return true if (graph is directed, false otherwise.

Public Members

size_t **_num_of_edges** = 0

nodeview_t **_node**

graph_attr_dict_factory **graph** = { }

adjlist_outer_dict_factory **_adj**

Public Static Functions

static inline *edge_t* &**end_points**(*edge_t* &e)

For compatible with BGL adaptor.

Parameters **e** – **[in]**

Returns edge_t&

static inline const *edge_t* &**end_points**(const *edge_t* &e)

For compatible with BGL adaptor.

Parameters **e** – **[in]**

Returns edge_t&

Template Class NodeView

- Defined in file_xnetwork_classes_reportviews.hpp

Page Contents

Template Parameter Order

1. `typename nodeview_t`

Class Documentation

template<typename **nodeview_t**>

class **NodeView**

A *NodeView* class to act as G.nodes for a XNetwork *Graph* Set operations act on the nodes without considering data. Iteration is over nodes. Node data can be looked up like a dict. Use NodeDataView to iterate over node data or to specify a data attribute for lookup. NodeDataView is created by calling the *NodeView*.

Parameters

graph : XNetwork graph-like class

Examples

G = xn::path_graph(3); NV = G.nodes(); 2 : NV

true

for n : NV: print(n);

0 1 2

assert(NV & {1, 2, 3} == {1, 2}); G.add_node(2, color="blue"); NV[2];

{"color": "blue"}

G.add_node(8, color="red"); NDV = G.nodes(data=true); (2, NV[2]] : NDV

true

for n, dd : NDV: print((n, dd.get("color", "aqua")));

(0, "aqua"); (1, "aqua"); (2, "blue"); (8, "red");

NDV[2] == NV[2];

true

NVdata = G.nodes(data="color", default="aqua"); (2, NVdata[2]] : NVdata

true

for n, dd : NVdata: print((n, dd));

(0, "aqua"); (1, "aqua"); (2, "blue"); (8, "red");

NVdata[2] == NV[2]; // NVdata gets "color", NV gets datadict

false

Public Functions

inline explicit **NodeView**(*nodeview_t* &nodes)

inline auto **size**()

inline auto **begin**()

inline auto **end**()

inline auto **operator[]**(const Node &n)

inline bool **contains**(const Node &n)

Template Class VertexView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::Graph< __nodeview_t, adjlist_t >` (*Template Class Graph*)

Derived Type

- `public xn::grAdaptor< Graph >` (*Template Class grAdaptor*)

Template Parameter Order

1. **`Graph <exhale_class_classxn_1_1Graph_>`_**

Class Documentation

template<typename **Graph**>

class **VertexView** : public xn::*Graph*<__nodeview_t, adjlist_t>

Template Parameters **Graph** –

Subclassed by *xn::grAdaptor< Graph >*

Public Functions

inline explicit **VertexView**(*Graph* &&G) noexcept

Construct a new Vertex View object.

Parameters **G** – **[in]**

inline auto **begin**() const

Returns auto

inline auto **end**() const

Returns auto

inline auto **cbegin**() const

Returns auto

inline auto **cend**() const

Returns auto

1.4.3 Enums

Enum CUTStatus

- Defined in file_ellcpp_cut_config.hpp

Enum Documentation

enum **CUTStatus**

Values:

enumerator **success**

enumerator **nosoln**

enumerator **smallenough**

enumerator **noeffect**

Enum CUTSTATUS

- Defined in file_ellip_ellipsoid.hpp

Enum Documentation

enum **CUTSTATUS**

Return return status

Values:

enumerator **CUT**

enumerator **NOSOLUTION**

enumerator **NOEFFECT**

Enum STATUS

- Defined in file_ellip_ellipsoid.hpp

Enum Documentation

enum **STATUS**

Return return status

Values:

enumerator **FOUND**

enumerator **EXCEEDMAXITER**

enumerator **NOTFOUND**

1.4.4 Functions

Template Function algo::half_nonnegative

- Defined in file_ellcpp_half_nonnegative.hpp

Function Documentation

template<typename **N**>
auto `algo::half_nonnegative(`*N* `n)` noexcept -> typename std::enable_if<std::is_integral<*N*>::value, *N*>::type

Template Function bsearch

- Defined in file_ellcpp_cutting_plane.hpp

Function Documentation

template<typename **Oracle**, typename **Space**>
auto **bsearch**(*Oracle* &&Omega, *Space* &&I, const *Options* &options = *Options*()) -> *CInfo*

Template Parameters

- **Oracle** –
- **Space** –

Parameters

- **Omega** – **[inout]** perform assessment on x0
- **I** – **[inout]** interval containing x*
- **options** – **[in]** maximum iteration and error tolerance etc.

Returns *CInfo*

Template Function cutting_plane_dc

- Defined in file_ellcpp_cutting_plane.hpp

Function Documentation

template<typename **Oracle**, typename **Space**, typename **opt_type**>
auto **cutting_plane_dc**(*Oracle* &&Omega, *Space* &&S, *opt_type* &&t, const *Options* &options = *Options*())

Cutting-plane method for solving convex problem.

Template Parameters

- **Oracle** –
- **Space** –
- **opt_type** –

Parameters

- **Omega** – **[inout]** perform assessment on x0
- **S** – **[inout]** search Space containing x*
- **t** – **[inout]** best-so-far optimal sol'n
- **options** – **[in]** maximum iteration and error tolerance etc.

Returns Information of Cutting-plane method

Template Function cutting_plane_feas

- Defined in file_ellcpp_cutting_plane.hpp

Function Documentation

template<typename **Oracle**, typename **Space**>
auto **cutting_plane_feas**(*Oracle* &&Omega, *Space* &&S, const *Options* &options = *Options*()) -> *CInfo*

Find a point in a convex set (defined through a cutting-plane oracle).

A function f(x) is *convex* if there always exist a g(x) such that f(z) >= f(x) + g(x)' * (z - x), forall z, x in dom f. Note that dom f does not need to be a convex set in our definition. The affine function g' (x - xc) + beta is called a cutting-plane, or a ``cut" for short. This algorithm solves the following feasibility problem:

```
find x
s.t. f(x) <= 0,
```

A *separation oracle* asserts that an evalution point x0 is feasible, or provide a cut that separates the feasible region and x0.

Template Parameters

- **Oracle** –
- **Space** –

Parameters

- **Omega** – **[inout]** perform assessment on x0
- **S** – **[inout]** search Space containing x*
- **options** – **[in]** maximum iteration and error tolerance etc.

Returns Information of Cutting-plane method

Template Function cutting_plane_q

- Defined in file_ellcpp_cutting_plane.hpp

Function Documentation

template<typename **Oracle**, typename **Space**, typename **opt_type**>
auto **cutting_plane_q**(*Oracle* &&Omega, *Space* &&S, *opt_type* &&t, const *Options* &options = *Options*())

Cutting-plane method for solving convex discrete optimization problem.

Cutting-plane method for solving convex discrete optimization problem input oracle perform assessment on x0 S(xc) Search space containing x* t best-so-far optimal sol'n max_it maximum number of iterations tol error tolerance output x solution vector niter number of iterations performed

Template Parameters

- **Oracle** –
- **Space** –

Parameters

- **Omega** – **[inout]** perform assessment on x0
- **S** – **[inout]** search Space containing x*
- **t** – **[inout]** best-so-far optimal sol'n
- **options** – **[in]** maximum iteration and error tolerance etc.

Returns Information of Cutting-plane method

Template Function ellipsoid_algo

- Defined in file_ellip_ellipsoid.hpp

Function Documentation

template<class **Enclosing**, class **Oracle**, class **Vec**>
STATUS **ellipsoid_algo**(*Enclosing* &E, *Oracle* &P, *Vec* &best_x, double &best_f, int max_it = 100, double tol = 1e-4)

Template Function ellipsoid_dc

- Defined in file_ellip_ellipsoid.hpp

Function Documentation

template<class **Enclosing**, class **Oracle**, class **Vec**>
STATUS **ellipsoid_dc**(*Enclosing* &E, *Oracle* &P, *Vec* &best_x, double &best_f, int max_it = 100, double tol = 1e-4)

Template Function ellipsoid_dc_discrete

- Defined in file_ellip_ellipsoid.hpp

Function Documentation

template<class **Enclosing**, class **Oracle**, class **Vec**>
STATUS **ellipsoid_dc_discrete**(*Enclosing* &E, *Oracle* &P, *Vec* &best_x, double &best_f, int max_it = 100, double tol = 1e-4)

Template Function eval

- Defined in file_ellip_rgp_yalaa.cpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "eval" with arguments (const AF&, const pmap&) in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml. Potential matches:

```
- auto eval(const edge_t &e, const Arr &x) const -> double
- auto eval(const edge_t &e, const double &x) const -> double
```

Template Function fun::gcd(_Mn, _Mn)

- Defined in file_py2cpp_fractions-new.hpp

Function Documentation

template<typename **_Mn**>
constexpr *_Mn* fun::**gcd**(*_Mn* __m, *_Mn* __n)

Greatest common divider.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**

- **__n** – **[in]**

Returns _Mn

Template Function fun::gcd(Mn, Mn)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Mn**>
constexpr auto `fun::gcd`(*Mn* _m, *Mn* _n) -> *Mn*

Greatest common divider.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**
- **__n** – **[in]**

Returns _Mn

Template Function fun::lcm(_Mn, _Mn)

- Defined in file_py2cpp_fractions-new.hpp

Function Documentation

template<typename **_Mn**>
constexpr *_Mn* `fun::lcm`(*_Mn* __m, *_Mn* __n)

Least common multiple.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**
- **__n** – **[in]**

Returns _Mn

Template Function fun::lcm(Mn, Mn)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Mn**>
constexpr auto `fun::lcm`(*Mn* _m, *Mn* _n) -> *Mn*

Least common multiple.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**
- **__n** – **[in]**

Returns _Mn

Template Function fun::operator*

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Z**>
constexpr auto `fun::operator*`(int &&c, const *Fraction*<*Z*> &frac) -> *Fraction*<*Z*>

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Template Function fun::operator+(const Z&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Z**>
constexpr auto `fun::operator+`(const *Z* &c, const *Fraction*<*Z*> &frac) -> *Fraction*<*Z*>

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Template Function fun::operator+(int&&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Z**>
constexpr auto `fun::operator+`(int &&c, const *Fraction*<*Z*> &frac) -> *Fraction*<*Z*>

Parameters

- **c** – **[in]**
- **frac** – **[in]**
- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Returns Fraction<Z>

Template Function fun::operator-(const Z&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Z**>
constexpr auto `fun::operator-`(const *Z* &c, const *Fraction*<*Z*> &frac) -> *Fraction*<*Z*>

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Template Function fun::operator-(int&&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Z**>
constexpr auto `fun::operator-`(int &&c, const *Fraction*<*Z*> &frac) -> *Fraction*<*Z*>

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

Template Function fun::operator<<

- Defined in file_py2cpp_fractions.hpp

Function Documentation

template<typename **Stream**, typename **Z**>
auto **fun::operator<<**(*Stream* &os, const *Fraction*<*Z*> &frac) -> *Stream*&

Template Parameters

- **_Stream** –
- **Z** –

Parameters

- **os** – **[in]**
- **frac** – **[in]**

Returns _Stream&

Function main()

- Defined in file_ellip_micp_test1.cpp

Function Documentation

Warning: doxygenfunction: Cannot find function "main" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Function main()

- Defined in file_ellip_profitmaxprob.cpp

Function Documentation

Warning: doxygenfunction: Cannot find function "main" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Function main()

- Defined in file_ellip_profitmaxprob_2.cpp

Function Documentation

Warning: doxygenfunction: Cannot find function "main" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Template Function max

- Defined in file_ellip_rgp_yalaa.cpp

Function Documentation

Warning: doxygenfunction: Cannot find function "max" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Template Function max_parametric

- Defined in file_netoptim_parametric.hpp

Function Documentation

template<typename **Graph**, typename **T**, typename **Fn1**, typename **Fn2**, typename **Container**>
auto **max_parametric**(const *Graph* &G, *T* &r_opt, *Fn1* &&d, *Fn2* &&zero_cancel, *Container* &&dist, size_t max_iter = 1000)

maximum parametric problem

This function solves the following network parametric problem:

```
max  r
s.t. dist[v] - dist[u] \ge d(u, v, r)
     \forall e(u, v) \in G(V, E)
```

Template Parameters

- **Graph** –
- **T** –
- **Fn1** –
- **Fn2** –
- **Container** –

Parameters

- **G** – **[in]** directed graph
- **r_opt** – **[inout]** parameter to be maximized, initially a large number
- **d** – **[in]** monotone decreasing function w.r.t. r
- **zero_cancel** – **[in]**

- **dist** – **[inout]**

Returns optimal r and the critical cycle

Template Function min_cycle_ratio

- Defined in file_netoptim_min_cycle_ratio.hpp

Function Documentation

template<typename **Graph**, typename **T**, typename **Fn1**, typename **Fn2**, typename **Container**>
auto **min_cycle_ratio**(const *Graph* &G, *T* &r0, *Fn1* &&get_cost, *Fn2* &&get_time, *Container* &&dist, size_t max_iter = 1000)

minimum cost-to-time cycle ratio problem

This function solves the following network parametric problem:

```
max  r
s.t. dist[v] - dist[u] \ge cost(u, v) - r * time(u, v)
     \forall e(u, v) \in G(V, E)
```

Template Parameters

- **Graph** –
- **Fn1** –
- **Fn2** –
- **Container** –

Parameters

- **G** – **[in]**
- **r0** – **[inout]**
- **get_cost** – **[in]**
- **get_time** – **[in]**
- **dist** – **[inout]**

Returns auto

Template Function norm

- Defined in file_ellip_ellipsoid.hpp

Function Documentation

template<class **Vec**>
inline double **norm**(const *Vec* &x)

Template Function operator*(const _Up&, const monomial<_Tp>&)

- Defined in file_ellip_monomial.hpp

Function Documentation

template<typename **_Tp**, typename **_Up**>
inline *monomial*<*_Tp*> **operator***(const *_Up* &c, const *monomial*<*_Tp*> &m)

Multiplication

Template Function operator*(monomial<_Tp>, const monomial<_Tp>&)

- Defined in file_ellip_monomial.hpp

Function Documentation

template<typename **_Tp**>
inline *monomial*<*_Tp*> **operator***(*monomial*<*_Tp*> lhs, const *monomial*<*_Tp*> &rhs)

Multiply

Template Function operator*(posynomial<_Tp>, const monomial<_Tp>&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator***(*posynomial*<*_Tp*> p, const *monomial*<*_Tp*> &m)

Multiply

Template Function operator*(posynomial<_Tp>, const _Tp&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator***(*posynomial*<*_Tp*> p, const *_Tp* &c)

Multiply

Template Function operator+(const monomial<_Tp>&, const monomial<_Tp>&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator+**(const *monomial*<*_Tp*> &m1, const *monomial*<*_Tp*> &m2)

Add

Template Function operator+(posynomial<_Tp>, const monomial<_Tp>&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator+**(*posynomial*<*_Tp*> p, const *monomial*<*_Tp*> &m)

Add

Template Function operator/(const _Up&, const monomial<_Tp>&)

- Defined in file_ellip_monomial.hpp

Function Documentation

template<typename **_Tp**, typename **_Up**>
inline *monomial*<*_Tp*> **operator/**(const *_Up* &c, const *monomial*<*_Tp*> &m)

Division

Template Function operator/(monomial<_Tp>, const monomial<_Tp>&)

- Defined in file_ellip_monomial.hpp

Function Documentation

template<typename **_Tp**>
inline *monomial*<*_Tp*> **operator/**(*monomial*<*_Tp*> lhs, const *monomial*<*_Tp*> &rhs)

Divide

Template Function operator/(posynomial<_Tp>, const monomial<_Tp>&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator/**(*posynomial*<*_Tp*> p, const *monomial*<*_Tp*> &m)

Divide

Template Function operator/(posynomial<_Tp>, const _Tp&)

- Defined in file_ellip_posynomial.hpp

Function Documentation

template<typename **_Tp**>
posynomial<*_Tp*> **operator/**(*posynomial*<*_Tp*> p, const *_Tp* &c)

Divide

Template Function py::enumerate

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "py::enumerate" with arguments (T&&) in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename T, typename TIter = decltype(std::begin(std::declval<T>())),␣
↪typename = decltype(std::end(std::declval<T>()))> constexpr auto enumerate(T &&
↪iterable)
```

Template Function py::len(const set<Key>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **Key**>
inline auto `py::len`(const *set*<*Key*> &m) -> size_t

Template Parameters **Key** –

Parameters **m** – **[in]**

Returns size_t

Template Function py::len(const dict<Key, T>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **Key**, typename **T**>
inline auto `py::len`(const *dict*<*Key*, *T*> &m) -> size_t

Template Parameters

- **Key** –
- **T** –

Parameters **m** – **[in]**

Returns size_t

Template Function py::operator<(const Key&, const set<Key>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **Key**>
inline auto `py::operator<`(const *Key* &key, const *set*<*Key*> &m) -> bool

Template Parameters **Key** –

Parameters

- **key** – **[in]**
- **m** – **[in]**

Returns true

Returns false

Template Function py::operator<(const Key&, const dict<Key, T>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **Key**, typename **T**>
inline auto **py::operator<**(const *Key* &key, const *dict*<*Key*, *T*> &m) -> bool

Template Parameters

- **Key** –
- **T** –

Parameters

- **key** – **[in]**
- **m** – **[in]**

Returns true

Returns false

Template Function py::range(T, T)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **T**>
inline constexpr auto **py::range**(*T* start, *T* stop)

Template Function py::range(T)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **T**>
inline constexpr auto **py::range**(*T* stop)

Template Function sqrt

- Defined in file_ellip_monomial.hpp

Function Documentation

template<typename **_Tp**>
inline *monomial*<*_Tp*> **sqrt**(*monomial*<*_Tp*> m)

Square root

Template Function zeros(std::initializer_list<T>&&)

- Defined in file_ellcpp_utility.hpp

Function Documentation

template<typename **T**>
constexpr auto **zeros**(std::initializer_list<*T*> &&x) -> typename std::enable_if<std::is_integral<*T*>::value, Arr>::type

Template Function zeros(const T&)

- Defined in file_ellcpp_utility.hpp

Function Documentation

template<typename **T**>
constexpr auto **zeros**(const *T*&) noexcept(noexcept(*T*{})) -> typename std::enable_if<std::is_floating_point<*T*>::value, *T*>::type

1.4.5 Variables

Variable e1

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e1" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable e2

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e2" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable e3

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e3" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable e4

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e4" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable e5

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e5" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable e6

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "e6" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable ui

- Defined in file_ellip_profitmaxprob_2.cpp

Variable Documentation

Warning: doxygenvariable: Cannot find variable "ui" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Variable xn::__slots__

- Defined in file_xnetwork_classes_reportviews.hpp

Variable Documentation

static const auto `xn::__slots__` = ()

A DataView class for nodes of a XNetwork *Graph*

The main use for this class is to iterate through node-data pairs. The data can be the entire data-dictionary for each node, or it can be a specific attribute (with default) for each node. Set operations are enabled with NodeDataView, but don't work in cases where the data is not hashable. Use with caution. Typically, set operations on nodes use *NodeView*, not NodeDataView. That is, they use `G.nodes` instead of `G.nodes(data="foo")`.

Parameters

graph : XNetwork graph-like class data : bool or string (default=false); default : object (default=None);

```
A View class for degree of nodes : a XNetwork Graph
```

The functionality is like dict.items() with (node, degree) pairs. Additional functionality includes read-only lookup of node degree, and calling with optional features nbunch (for only a subset of nodes); and weight (use edge weights to compute degree).

Parameters

graph : XNetwork graph-like class nbunch : node, container of nodes, or None meaning all nodes (default=None); weight : bool or string (default=None);

Notes

DegreeView can still lookup any node even if (nbunch is specified.

Examples

G = xn::path_graph(3); DV = G.degree(); assert(DV[2] == 1); assert(sum(deg for n, deg : DV) == 4);

DVweight = G.degree(weight="span"); G.add_edge(1, 2, span=34); DVweight[2];

34

DVweight[0]; // default edge weight is 1

1

sum(span for n, span : DVweight); // sum weighted degrees

70

DVnbunch = G.degree(nbunch=(1, 2)); assert(len(list(DVnbunch)) == 2); // iteration over nbunch only

```
A DegreeView class to act as G.degree for a XNetwork Graph
```

Typical usage focuses on iteration over `(node, degree)` pairs. The degree is by default the number of edges incident to the node. Optional argument `weight` enables weighted degree using the edge attribute named : the `weight` argument. Reporting and iteration can also be restricted to a subset of nodes using `nbunch`.

Additional functionality include node lookup so that `G.degree[n]` reported the (possibly weighted) degree of node `n`. Calling the view creates a view with different arguments `nbunch` or `weight`.

Parameters

graph : XNetwork graph-like class nbunch : node, container of nodes, or None meaning all nodes (default=None); weight : string or None (default=None);

Notes

DegreeView can still lookup any node even if (nbunch is specified.

Examples

G = xn::path_graph(3); DV = G.degree(); assert(DV[2] == 1); assert(G.degree[2] == 1); assert(sum(deg for n, deg : DV) == 4);

DVweight = G.degree(weight="span"); G.add_edge(1, 2, span=34); DVweight[2];

34

DVweight[0]; // default edge weight is 1

1

sum(span for n, span : DVweight); // sum weighted degrees

70

DVnbunch = G.degree(nbunch=(1, 2)); assert(len(list(DVnbunch)) == 2); // iteration over nbunch only

A DegreeView class to report out_degree for a DiGraph; See DegreeView

A DegreeView class to report in_degree for a DiGraph; See DegreeView

A DegreeView class for undirected multigraphs; See DegreeView

A DegreeView class for MultiDiGraph; See DegreeView

A DegreeView class for inward degree of MultiDiGraph; See DegreeView

A DegreeView class for outward degree of MultiDiGraph; See DegreeView

EdgeDataView for outward edges of DiGraph; See EdgeDataView

```
A EdgeDataView class for edges of Graph
```

This view is primarily used to iterate over the edges reporting edges as node-tuples with edge data optionally reported. The argument `nbunch` allows restriction to edges incident to nodes : that container/singleton. The default (nbunch=None); reports all edges. The arguments `data` and `default` control what edge data is reported. The default `data == false` reports only node-tuples for each edge. If `data is true` the entire edge data dict is returned. Otherwise `data` is assumed to hold the name of the edge attribute to report with default `default` if (that edge attribute is not present.

Parameters

nbunch : container of nodes, node or None (default None); data : false, true or string (default false); default : default value (default None);

Examples

G = xn::path_graph(3); G.add_edge(1, 2, foo="bar"); list(G.edges(data="foo", default="biz"));

[(0, 1, "biz"), (1, 2, "bar")];

assert((0, 1, "biz"] : G.edges(data="foo", default="biz"));

An EdgeDataView class for outward edges of DiGraph; See EdgeDataView

An EdgeDataView for outward edges of MultiDiGraph; See EdgeDataView

1.4.6 Defines

Define _PROFIX_MAX_HPP

- Defined in file_ellip_profitmaxprob.hpp

Define Documentation

_PROFIX_MAX_HPP

Define ELL_LIKELY

- Defined in file_ellcpp_ell_assert.hpp

Define Documentation

ELL_LIKELY(x)

Define ELL_UNLIKELY

- Defined in file_ellcpp_ell_assert.hpp

Define Documentation

ELL_UNLIKELY(x)

1.4.7 Typedefs

Typedef aaf

- Defined in file_ellip_profitmaxprob_2.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "aaf" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef aaf

- Defined in file_ellip_rgp_yalaa.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "aaf" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef Arr

- Defined in file_ellcpp_utility.hpp

Typedef Documentation

using *ell*::**Arr** = xt::xarray<double, xt::layout_type::row_major>

Typedef iv_t

- Defined in file_ellip_rgp_yalaa.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "iv_t" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef iv_traits

- Defined in file_ellip_rgp_yalaa.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "iv_traits" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef ivt

- Defined in file_ellip_profitmaxprob_2.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "ivt" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef pmap

- Defined in file_ellip_rgp_yalaa.cpp

Typedef Documentation

Warning: doxygentypedef: Cannot find typedef "pmap" in doxygen xml output for project "ellcpp" from directory: ./doxyoutput/xml

Typedef Value_type

- Defined in file_py2cpp_py2cpp.hpp

Typedef Documentation

using **Value_type** = typename T::value_type

Typedef Vec

- Defined in file_ellip_gp_solve.cpp

Typedef Documentation

using *ellipsoid*::**Vec** = std::valarray<double>

Typedef Vec

- Defined in file_ellip_rgp_yalaa.cpp

Typedef Documentation

using *ellipsoid*::**Vec** = std::valarray<double>

Typedef xn::SimpleDiGraphS

- Defined in file_xnetwork_classes_digraphs.hpp

Typedef Documentation

using xn::**SimpleDiGraphS** = *DiGraphS*<decltype(py::*range*<int>(1)), py::*dict*<int, int>>

Typedef xn::SimpleGraph

- Defined in file_xnetwork_classes_graph.hpp

Typedef Documentation

using xn::**SimpleGraph** = *Graph*<decltype(py::*range*<int>(1)), py::*set*<int>>

You read all the way to the bottom?! This text is specified by giving an argument to `afterBodySummary`. As the docs state, this summary gets put in after a **lot** of information. It's available for you to use if you want it, but from a design perspective it's rather unlikely any of your users will even see this text.

CHAPTER

TWO

HOW THIS VERSION OF ELLCPP WAS CREATED

For convenience, I'm going to inline the code used in this configuration from `conf.py` here. The three main things you need to do here are

1. The `requirements.txt` used on read the docs.
2. Setup the `breathe` and `exhale` extensions.
3. Choose your `html_theme`, which affects what you choose for the `exhale` side.

Refer to the Start to finish for Read the Docs tutorial for getting everything setup on RTD.

2.1 requirements.txt

```
# for testing the master branch
# git+git://github.com/svenevs/exhale.git#egg=exhale
# See: https://exhale.readthedocs.io/en/latest/#exhale-version-compatibility-with-python-
↪sphinx-and-breathe
sphinx>=2.0
sphinx-bootstrap-theme>=0.4.0
breathe>=4.13.0
exhale
```

2.2 Extension Setup

```
# Tell Sphinx to use both the `breathe` and `exhale` extensions
extensions = [
    'breathe',
    'exhale'
]

# Setup the `breathe` extension
breathe_projects = {"ellcpp": "./doxyoutput/xml"}
breathe_default_project = "ellcpp"

# Setup the `exhale` extension
# import textwrap
exhale_args = {
    ############################################################################
```

(continues on next page)

(continued from previous page)

```
    # These arguments are required.                                          #
    ##########################################################################
    "containmentFolder":     "./api",
    "rootFileName":          "library_root.rst",
    "rootFileTitle":         "Library API",
    "doxygenStripFromPath":  "../lib/include",
    ##########################################################################
    # Suggested optional arguments.                                          #
    ##########################################################################
    "createTreeView":        True,
    "exhaleExecutesDoxygen": True,
    "exhaleDoxygenStdin": textwrap.dedent('''
        INPUT       = ../lib/include
        # For this code-base, the following helps Doxygen get past a macro
        # that it has trouble with.  It is only meaningful for this code,
        # not for yours.
        PREDEFINED += NAMESPACE_BEGIN(arbitrary)="namespace arbitrary {"
        PREDEFINED += NAMESPACE_END(arbitrary)="}"
    '''),
    ##########################################################################
    # HTML Theme specific configurations.                                    #
    ##########################################################################
    # Fix broken Sphinx RTD Theme 'Edit on GitHub' links
    # Search for 'Edit on GitHub' on the FAQ:
    #     http://exhale.readthedocs.io/en/latest/faq.html
    "pageLevelConfigMeta": ":github_url: https://github.com/svenevs/exhale-companion",
    ##########################################################################
    # Main library page layout example configuration.                        #
    ##########################################################################
    "afterTitleDescription": textwrap.dedent(u'''
        Welcome to the developer reference to Exhale Companion.  The code being
        documented here is largely meaningless and was only created to test
        various corner cases e.g. nested namespaces and the like.

        .. note::

            The text you are currently reading was fed to ``exhale_args`` using
            the :py:data:`~exhale.configs.afterTitleDescription` key.  Full
            reStructuredText syntax can be used.

        .. tip::

           Sphinx / Exhale support unicode!  You're ``conf.py`` already has
           it's encoding declared as ``# -*- coding: utf-8 -*-`` **by
           default**.  If you want to pass Unicode strings into Exhale, simply
           prefix them with a ``u`` e.g. ``u""`` (of course you would
           actually do this because you are writing with åçćëñtß or
           non-English  ).
    '''),
    "afterHierarchyDescription": textwrap.dedent('''
        Below the hierarchies comes the full API listing.
```

(continues on next page)

(continued from previous page)

```
        1. The text you are currently reading is provided by
           :py:data:`~exhale.configs.afterHierarchyDescription`.
        2. The Title of the next section *just below this* normally defaults to
           ``Full API``, but the title was changed by providing an argument to
           :py:data:`~exhale.configs.fullApiSubSectionTitle`.
        3. You can control the number of bullet points for each linked item on
           the remainder of the page using
           :py:data:`~exhale.configs.fullToctreeMaxDepth`.
    '''),
    "fullApiSubSectionTitle": "Custom Full API SubSection Title",
    "afterBodySummary": textwrap.dedent('''
        You read all the way to the bottom?!  This text is specified by giving
        an argument to :py:data:`~exhale.configs.afterBodySummary`.  As the docs
        state, this summary gets put in after a **lot** of information.  It's
        available for you to use if you want it, but from a design perspective
        it's rather unlikely any of your users will even see this text.
    '''),
    ##############################################################################
    # Individual page layout example configuration.                              #
    ##############################################################################
    # Example of adding contents directives on custom kinds with custom title
    "contentsTitle": "Page Contents",
    "kindsWithContentsDirectives": ["class", "file", "namespace", "struct"],
    # This is a testing site which is why I'm adding this
    "includeTemplateParamOrderList": True,
    ##############################################################################
    # useful to see ;)
    "verboseBuild": True
}

# Tell sphinx what the primary language being documented is.
primary_domain = 'cpp'

# Tell sphinx what the pygments highlight language should be.
highlight_language = 'cpp'
```

2.3 HTML Theme Setup

```
# The name of the Pygments (syntax highlighting) style to use.
# `sphinx` works very well with the RTD theme, but you can always change it
pygments_style = 'sphinx'

# on_rtd is whether we are on readthedocs.org, this line of code grabbed from docs.
↪readthedocs.org
on_rtd = os.environ.get('READTHEDOCS', None) == 'True'

if not on_rtd:  # only import and set the theme if we're building docs locally
    import sphinx_bootstrap_theme
    html_theme = 'bootstrap'
    html_theme_path = sphinx_bootstrap_theme.get_html_theme_path()
```

CHAPTER

THREE

USING INTERSPHINX

The Sphinx intersphinx extension is exceptionally convenient, and typically works out-of-the-box for most projects you would want to link to. This is not limited to linking to documents just within your domain, and if you really want to go the extra mile (and create your own mapping), it doesn't even have to be restricted to linking to documentation that was generated with Sphinx.

Contents

3.1 Setup your `conf.py`

First, how you link to things depends on what your domain is. In the Exhale Quickstart Guide, I encouraged you to add these lines to your `conf.py`:

```
# Tell sphinx what the primary language being documented is.
primary_domain = 'cpp'

# Tell sphinx what the pygments highlight language should be.
highlight_language = 'cpp'
```

This will come up in the next section, but is added to `conf.py` so it is included here.

For this ellcpp project, I want to link to two Sphinx generated projects. In the `conf.py`, this means that I have:

```
# In addition to `breathe` and `exhale`, use the `intersphinx` extension
extensions = [
    'sphinx.ext.intersphinx',
    'breathe',
    'exhale'
```

(continues on next page)

(continued from previous page)

```
]

# Specify the baseurls for the projects I want to link to
intersphinx_mapping = {
    'exhale':  ('https://exhale.readthedocs.io/en/latest/', None),
    'nanogui': ('http://nanogui.readthedocs.io/en/latest/', None)
}
```

3.2 Linking to Other Sites Using Intersphinx

This is where understanding your primary domain becomes particularly relevant. Since the `primary_domain` for this project is `cpp`, I can link to things like `:cpp:function:` as just `:function:`. But if I want to link to Python or C domains I need to specify that explicitly. Inlined from the Cross Referencing Syntax docs, there is some syntax you will likely need to wield:

- You may supply an explicit title and reference target: `:role:`title <target>` will refer to target, but the link text will be title.
- If you prefix the content with `!`, no reference/hyperlink will be created.
- If you prefix the content with `~`, the link text will only be the last component of the target. For example, `:py:meth:`~Queue.Queue.get` will refer to `Queue.Queue.get` but only display `get` as the link text.

3.2.1 Linking to Python Docs from a `cpp` Project

Since I've setup intersphinx to point back to the main Exhale site, I'll just link to some from there.

Linking to a Python Class

**`:py:class:`exhale.graph.ExhaleRoot`** Links to `exhale.graph.ExhaleRoot`

**`:py:class:`graph.ExhaleRoot <exhale.graph.ExhaleRoot>`** Links to `graph.ExhaleRoot`

**`:py:class:`~exhale.graph.ExhaleRoot`** Links to `ExhaleRoot`

Linking to a Python Function

**`:py:func:`exhale.deploy.explode`** Links to `exhale.deploy.explode()`

**`:py:func:`deploy.explode <exhale.deploy.explode>`** Links to `deploy.explode`

**`:py:func:`~exhale.deploy.explode`** Links to `explode()`

3.2.2 Linking to Another C++ Project

This is where understanding how to manipulate the link titles becomes relevant. I'll use the NanoGUI docs since I stole the `NAMESPACE_BEGIN` macro from there.

Linking to a C++ Class Using a single `:` does not appear to work, but using the `namespace::ClassName` seems to include a leading `:`. I think this is a bug, but solving it would likely be treacherous so instead just control the title yourself.

**`:class:`nanogui::Screen`** Links to `nanogui::Screen`

**`:class:`nanogui::Screen <nanogui::Screen>`** Links to `nanogui::Screen`

:class:`~nanogui::Screen` Links to Screen

Linking to C Domains Even if the other project is primarily C++, things like macros are in the `:c:` Sphinx domain. I choose the `NAMESPACE_BEGIN` example to show you how to qualify where Sphinx should link — both **this project** and **NanoGUI** have links to it, so when I just do `:c:macro:`NAMESPACE_BEGIN` the link (NAMESPACE_BEGIN) goes to **this project**. Using `nanogui:NAMESPACE_BEGIN` (since `'nanogui'` was a key in our `intersphinx_mapping`)

:c:macro:`nanogui:NAMESPACE_BEGIN` Links to NAMESPACE_BEGIN

:c:macro:`NanoGUI macro NAMESPACE_BEGIN <nanogui:NAMESPACE_BEGIN>` Links to NanoGUI macro NAMESPACE_BEGIN

:c:macro:`~nanogui:NAMESPACE_BEGIN` Links to NAMESPACE_BEGIN

Tip: These kinds of cross references are **reStructuredText** syntax! You **must** enable the `\rst` environment for Doxygen (see Doxygen ALIASES) **and** use this in the documentation. For example, in order to get the NAMESPACE_BEGIN link to work, the actual C++ code is as follows:

```
#if !defined(NAMESPACE_BEGIN) || defined(DOXYGEN_DOCUMENTATION_BUILD)
    /**
     * \rst
     * See :c:macro:`NanoGUI macro NAMESPACE_BEGIN <nanogui:NAMESPACE_BEGIN>`.
     * \endrst
     */
    #define NAMESPACE_BEGIN(name) namespace name {
#endif
```

3.3 Finding the Links to Use

For things like classes that are qualified in namespaces, it should be pretty easy for you to figure out what the link is by inspection. However, there is an excellent tool available for you: the Sphinx Objects.inv Encoder/Decoder.

1. Install the utility:

```
$ pip install sphobjinv
```

2. Download the Sphinx `objects.inv` for the project you want to use. This should be at the location you specified in your `intersphinx_mapping`. So if the URL you gave was `url`, the `objects.inv` should be at `url/objects.inv`. Sticking with the NanoGUI example:

```
# Go to wherever you want and download the file
$ cd /tmp

# That's a capital 'Oh' not a zero; or use `wget`
$ curl -O http://nanogui.readthedocs.io/en/latest/objects.inv
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 44056  100 44056    0     0   109k      0 --:--:-- --:--:-- --:--:--  109k

# rename it so you know where it hails from
$ mv objects.inv nanogui_objects.inv
```

3. Decode it to plain text and search for what you are trying to link.

```
# decode it so we can search it
$ sphobjinv convert plain nanogui_objects.inv

Conversion completed.
'nanogui_objects.inv' decoded to 'nanogui_objects.txt'.

# search for the thing you are trying to link to
$ grep NAMESPACE_BEGIN nanogui_objects.txt | grep -v -- -1
                vvvvvvv
NAMESPACE_BEGIN c:macro 1 api/define_NAMESPACE_BEGIN.html#c.$ -
                ^^^^^^^
```

Tip: Refer to the sphobjinv syntax section, the reason I am piping to `grep -v -- -1` is because "priority" `-1` means it won't be available to link to. The `-v` tells `grep` to invert the match, and `--` tells `grep` that the command-line options (e.g., `-v`) are finished and what follows is an argument. That is, `-- -1` just makes it so `grep` doesn't think `-1` is a flag.

3.3.1 Custom Links

You can also make your own `intersphinx` mappings. I did this for linking to the BeautifulSoup docs. See the _intersphinx/README.md of Exhale.

This use case was for a dysfunctional `objects.inv`, but you could also easily create your own mapping to index a project that was not created using Sphinx.

3.4 Testing your Intersphinx Links

By default the Sphinx build process does not inform you of broken link targets when you run `make html`. The `sphinx-build` flag you want for testing this is `-n` (for *nitpicky*). You will want to make sure to `clean` first so that all errors get shown.

```
$ make SPHINXOPTS='-n' clean html
```

Tip: There is also a `make linkcheck` target for the Sphinx generated Makefiles!

Note: This was built using Exhale version 0.3.6.

Make sure to view the *Extension Setup* and *HTML Theme Setup* for the different versions, as they vary slightly (e.g., `bootstrap` gets more supplied in the `exhale_args` portion of `conf.py`).

INDEX

Symbols

A

B

C

E

F

G

I

L

M

N

O

P

Z