

---

# **graphslam Documentation**

*Release 0.0.7*

**Jeff Irion**

**Jan 13, 2022**



# CONTENTS

<b>1</b>	<b>graphslam</b>	<b>1</b>
1.1	graphslam package . . . . .	1
<b>2</b>	<b>Features</b>	<b>21</b>
<b>3</b>	<b>Installation</b>	<b>23</b>
<b>4</b>	<b>Example Usage</b>	<b>25</b>
4.1	SE(3) Dataset . . . . .	25
4.2	SE(2) Dataset . . . . .	26
<b>5</b>	<b>References and Links</b>	<b>29</b>
<b>6</b>	<b>Live Coding Graph SLAM in Python</b>	<b>31</b>
<b>7</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>
	<b>Index</b>	<b>37</b>



## GRAPHSLAM

### 1.1 graphslam package

#### 1.1.1 Subpackages

##### graphslam.edge package

##### Submodules

##### graphslam.edge.base\_edge module

A base class for edges.

**class** graphslam.edge.base\_edge.**BaseEdge** (*vertex\_ids, information, estimate, vertices=None*)  
Bases: object

A class for representing edges in Graph SLAM.

##### Parameters

- **vertex\_ids** (*list[int]*) – The IDs of all vertices constrained by this edge
- **information** (*np.ndarray*) – The information matrix  $\Omega_j$  associated with the edge
- **estimate** (*BasePose, np.ndarray, float*) – The expected measurement  $\mathbf{z}_j$
- **vertices** (*list[graphslam.vertex.Vertex], None*) – A list of the vertices constrained by the edge

##### **estimate**

The expected measurement  $\mathbf{z}_j$

**Type** *BasePose, np.ndarray, float*

##### **information**

The information matrix  $\Omega_j$  associated with the edge

**Type** *np.ndarray*

##### **vertex\_ids**

The IDs of all vertices constrained by this edge

**Type** *list[int]*

##### **vertices**

A list of the vertices constrained by the edge

**Type** list[*graphslam.vertex.Vertex*], None

**`_calc_jacobian`** (*err*, *dim*, *vertex\_index*)

Calculate the Jacobian of the edge with respect to the specified vertex's pose.

**Parameters**

- **`err`** (*np.ndarray*) – The current error for the edge (see *BaseEdge.calc\_error()*)
- **`dim`** (*int*) – The dimensionality of the compact pose representation
- **`vertex_index`** (*int*) – The index of the vertex (pose) for which we are computing the Jacobian

**Returns** The Jacobian of the edge with respect to the specified vertex's pose

**Return type** np.ndarray

**`calc_chi2`** ()

Calculate the  $\chi^2$  error for the edge.

$$\mathbf{e}_j^T \Omega_j \mathbf{e}_j$$

**Returns** The  $\chi^2$  error for the edge

**Return type** float

**`calc_chi2_gradient_hessian`** ()

Calculate the edge's contributions to the graph's  $\chi^2$  error, gradient (**b**), and Hessian (**H**).

**Returns**

- *float* – The  $\chi^2$  error for the edge
- *dict* – The edge's contribution(s) to the gradient
- *dict* – The edge's contribution(s) to the Hessian

**`calc_error`** ()

Calculate the error for the edge:  $\mathbf{e}_j \in \mathbb{R}^n$ .

**Returns** The error for the edge

**Return type** np.ndarray, float

**`calc_jacobians`** ()

Calculate the Jacobian of the edge's error with respect to each constrained pose.

$$\frac{\partial}{\partial \Delta \mathbf{x}^k} [\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)]$$

**Returns** The Jacobian matrices for the edge with respect to each constrained pose

**Return type** list[np.ndarray]

**`plot`** (*color=""*)

Plot the edge.

**Parameters** **`color`** (*str*) – The color that will be used to plot the edge

**`to_g2o`** ()

Export the edge to the .g2o format.

**Returns** The edge in .g2o format

**Return type** str

`graphsiam.edge.base_edge.EPSILON = 1e-06`  
 The difference that will be used for numerical differentiation

## graphsiam.edge.edge\_odometry module

A class for odometry edges.

**class** `graphsiam.edge.edge_odometry.EdgeOdometry` (*vertex\_ids, information, estimate, vertices=None*)

Bases: `graphsiam.edge.base_edge.BaseEdge`

A class for representing odometry edges in Graph SLAM.

### Parameters

- **vertices** (*list[graphsiam.vertex.Vertex]*) – A list of the vertices constrained by the edge
- **information** (*np.ndarray*) – The information matrix  $\Omega_j$  associated with the edge
- **estimate** (*BasePose*) – The expected measurement  $\mathbf{z}_j$

### vertices

A list of the vertices constrained by the edge

**Type** `list[graphsiam.vertex.Vertex]`

### information

The information matrix  $\Omega_j$  associated with the edge

**Type** `np.ndarray`

### estimate

The expected measurement  $\mathbf{z}_j$

**Type** `BasePose`

### calc\_error()

Calculate the error for the edge:  $\mathbf{e}_j \in \mathbb{R}^n$ .

$$\mathbf{e}_j = \mathbf{z}_j - (p_2 \ominus p_1)$$

**Returns** The error for the edge

**Return type** `np.ndarray`

### calc\_jacobians()

Calculate the Jacobian of the edge's error with respect to each constrained pose.

$$\frac{\partial}{\partial \Delta \mathbf{x}^k} [\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)]$$

**Returns** The Jacobian matrices for the edge with respect to each constrained pose

**Return type** `list[np.ndarray]`

### plot(color='b')

Plot the edge.

**Parameters** **color** (*str*) – The color that will be used to plot the edge

### to\_g2o()

Export the edge to the `.g2o` format.

**Returns** The edge in .g2o format

**Return type** str

## Module contents

### graphslam.pose package

#### Submodules

#### graphslam.pose.base\_pose module

A base class for poses.

**class** graphslam.pose.base\_pose.**BasePose**

Bases: numpy.ndarray

A base class for poses.

**copy** ()

Return a copy of the pose.

**Returns** A copy of the pose

**Return type** *BasePose*

**property** **inverse**

Return the pose's inverse.

**Returns** The pose's inverse

**Return type** *BasePose*

**jacobian\_boxplus** ()

Compute the Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = 0$ .

**Returns** The Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = 0$

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .



**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**property orientation**

Return the pose's orientation.

**Returns** The pose's orientation

**Return type** float, np.ndarray

**property position**

Return the pose's position.

**Returns** The pose's position

**Return type** np.ndarray

**to\_array** ()

Return the pose as a numpy array.

**Returns** The pose as a numpy array

**Return type** np.ndarray

`to_compact()`

Return the pose as a compact numpy array.

**Returns** The pose as a compact numpy array

**Return type** np.ndarray

## graphslam.pose.r2 module

Representation of a point in  $\mathbb{R}^2$ .

**class** graphslam.pose.r2.PoseR2

Bases: *graphslam.pose.base\_pose.BasePose*

A representation of a 2-D point.

**Parameters** `position` (*np.ndarray*, *list*) – The position in  $\mathbb{R}^2$

`copy()`

Return a copy of the pose.

**Returns** A copy of the pose

**Return type** *PoseR2*

**property** `inverse`

Return the pose's inverse.

**Returns** The pose's inverse

**Return type** *PoseR2*

`jacobian_boxplus()`

Compute the Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$ .

**Returns** The Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$

**Return type** np.ndarray

`jacobian_self_ominus_other_wrt_other(other)`

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** `other` (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

`jacobian_self_ominus_other_wrt_other_compact(other)`

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** `other` (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

`jacobian_self_ominus_other_wrt_self(other)`

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** `other` (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**jacobian\_self\_oplus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** `np.ndarray`

**jacobian\_self\_oplus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** `np.ndarray`

**jacobian\_self\_oplus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**jacobian\_self\_oplus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**property orientation**

Return the pose's orientation.

**Returns** A `PoseR2` object has no orientation, so this will always return 0.

**Return type** `float`

**property position**

Return the pose's position.

**Returns** The position portion of the pose

**Return type** `np.ndarray`

**to\_array** ()

Return the pose as a numpy array.

**Returns** The pose as a numpy array

**Return type** `np.ndarray`

**to\_compact** ()

Return the pose as a compact numpy array.

**Returns** The pose as a compact numpy array

**Return type** np.ndarray

### graphslam.pose.r3 module

Representation of a point in  $\mathbb{R}^3$ .

**class** graphslam.pose.r3.PoseR3

Bases: *graphslam.pose.base\_pose.BasePose*

A representation of a 3-D point.

**Parameters** *position* (*np.ndarray*, *list*) – The position in  $\mathbb{R}^3$

**copy** ()

Return a copy of the pose.

**Returns** A copy of the pose

**Return type** *PoseR3*

**property** *inverse*

Return the pose's inverse.

**Returns** The pose's inverse

**Return type** *PoseR3*

**jacobian\_boxplus** ()

Compute the Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$ .

**Returns** The Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** `other` (`BasePose`) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**`jacobian_self_oplus_other_wrt_other`** (`other`)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** `other` (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** `np.ndarray`

**`jacobian_self_oplus_other_wrt_other_compact`** (`other`)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** `other` (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** `np.ndarray`

**`jacobian_self_oplus_other_wrt_self`** (`other`)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** `other` (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**`jacobian_self_oplus_other_wrt_self_compact`** (`other`)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** `other` (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** `np.ndarray`

**`property orientation`**

Return the pose's orientation.

**Returns** A `PoseR3` object has no orientation, so this will always return 0.

**Return type** `float`

**`property position`**

Return the pose's position.

**Returns** The position portion of the pose

**Return type** `np.ndarray`

**`to_array`** ()

Return the pose as a numpy array.

**Returns** The pose as a numpy array

**Return type** `np.ndarray`

**`to_compact`** ()

Return the pose as a compact numpy array.

**Returns** The pose as a compact numpy array

**Return type** `np.ndarray`

## graphslam.pose.se2 module

Representation of a pose in  $SE(2)$ .

**class** graphslam.pose.se2.PoseSE2

Bases: *graphslam.pose.base\_pose.BasePose*

A representation of a pose in  $SE(2)$ .

### Parameters

- **position** (*np.ndarray*, *list*) – The position in  $\mathbb{R}^2$
- **orientation** (*float*) – The angle of the pose (in radians)

**copy** ()

Return a copy of the pose.

**Returns** A copy of the pose

**Return type** *PoseSE2*

**classmethod** **from\_matrix** (*matrix*)

Return the pose as an  $SE(2)$  matrix.

**Parameters** **matrix** (*np.ndarray*) – The  $SE(2)$  matrix that will be converted to a *PoseSE2* instance

**Returns** The matrix as a *PoseSE2* object

**Return type** *PoseSE2*

**property** **inverse**

Return the pose's inverse.

**Returns** The pose's inverse

**Return type** *PoseSE2*

**jacobian\_boxplus** ()

Compute the Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$ .

**Returns** The Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$

**Return type** *np.ndarray*

**jacobian\_self\_ominus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** *np.ndarray*

**jacobian\_self\_ominus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** *np.ndarray*

**jacobian\_self\_ominus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from *self*

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** **other** (*BasePose*) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** **other** (*BasePose*) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**property orientation**

Return the pose's orientation.

**Returns** The angle of the pose

**Return type** float

**property position**

Return the pose's position.

**Returns** The position portion of the pose

**Return type** np.ndarray

**to\_array** ()

Return the pose as a numpy array.

**Returns** The pose as a numpy array

**Return type** np.ndarray

`to_compact()`

Return the pose as a compact numpy array.

**Returns** The pose as a compact numpy array

**Return type** np.ndarray

`to_matrix()`

Return the pose as an  $SE(2)$  matrix.

**Returns** The pose as an  $SE(2)$  matrix

**Return type** np.ndarray

## graphslam.pose.se3 module

Representation of a pose in  $SE(3)$ .

**class** graphslam.pose.se3.PoseSE3

Bases: *graphslam.pose.base\_pose.BasePose*

A representation of a pose in  $SE(3)$ .

**Parameters**

- **position** (*np.ndarray, list*) – The position in  $\mathbb{R}^3$
- **orientation** (*np.ndarray, list*) – The orientation of the pose as a unit quaternion:  $[q_x, q_y, q_z, q_w]$

`copy()`

Return a copy of the pose.

**Returns** A copy of the pose

**Return type** *PoseSE3*

**property** `inverse`

Return the pose's inverse.

**Returns** The pose's inverse

**Return type** *PoseSE3*

`jacobian_boxplus()`

Compute the Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$ .

**Returns** The Jacobian of  $p_1 \boxplus \Delta x$  w.r.t.  $\Delta x$  evaluated at  $\Delta x = \mathbf{0}$

**Return type** np.ndarray

`jacobian_self_ominus_other_wrt_other(other)`

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

`jacobian_self_ominus_other_wrt_other_compact(other)`

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .

**Parameters** **other** (*BasePose*) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_2$ .



**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_ominus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being subtracted from `self`

**Returns** The Jacobian of  $p_1 \ominus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – (Unused) The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_other\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Parameters** *other* (`BasePose`) – (Unused) The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_2$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**jacobian\_self\_oplus\_other\_wrt\_self\_compact** (*other*)

Compute the Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Parameters** *other* (`BasePose`) – The pose that is being added to `self`

**Returns** The Jacobian of  $p_1 \oplus p_2$  w.r.t.  $p_1$ .

**Return type** np.ndarray

**normalize** ()

Normalize the quaternion portion of the pose.

**property orientation**

Return the pose's orientation.

**Returns** The pose's quaternion

**Return type** float

**property position**

Return the pose's position.

**Returns** The position portion of the pose

**Return type** np.ndarray

**to\_array()**

Return the pose as a numpy array.

**Returns** The pose as a numpy array

**Return type** np.ndarray

**to\_compact()**

Return the pose as a compact numpy array.

**Returns** The pose as a compact numpy array

**Return type** np.ndarray

**to\_matrix()**

Return the pose as an  $SE(3)$  matrix.

**Returns** The pose as an  $SE(3)$  matrix

**Return type** np.ndarray

## Module contents

### 1.1.2 Submodules

#### graphslam.graph module

A `Graph` class that stores the edges and vertices required for Graph SLAM.

Given:

- A set of  $M$  edges (i.e., constraints)  $\mathcal{E}$ 
  - $e_j \in \mathcal{E}$  is an edge
  - $\mathbf{e}_j \in \mathbb{R}^\bullet$  is the error associated with that edge, where  $\bullet$  is a scalar that depends on the type of edge
  - $\Omega_j$  is the  $\bullet \times \bullet$  information matrix associated with edge  $e_j$
- A set of  $N$  vertices  $\mathcal{V}$ 
  - $v_i \in \mathcal{V}$  is a vertex
  - $\mathbf{x}_i \in \mathbb{R}^c$  is the compact pose associated with  $v_i$
  - $\boxplus$  is the pose composition operator that yields a (non-compact) pose that lies in (a subspace of)  $\mathbb{R}^d$

We want to optimize

$$\chi^2 = \sum_{e_j \in \mathcal{E}} \mathbf{e}_j^T \Omega_j \mathbf{e}_j.$$

Let

$$\mathbf{x} := \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} \in \mathbb{R}^{cN}.$$

We will solve this optimization problem iteratively. Let

$$\mathbf{x}^{k+1} := \mathbf{x}^k \boxplus \Delta \mathbf{x}^k.$$

The  $\chi^2$  error at iteration  $k + 1$  is

$$\chi_{k+1}^2 = \sum_{e_j \in \mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^{k+1})]^T}_{1 \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\mathbf{e}_j(\mathbf{x}^{k+1})}_{\bullet \times 1}.$$

We will linearize the errors as:

$$\begin{aligned} \mathbf{e}_j(\mathbf{x}^{k+1}) &= \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k) \\ &\approx \mathbf{e}_j(\mathbf{x}^k) + \frac{\partial}{\partial \Delta \mathbf{x}^k} [\mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)] \Delta \mathbf{x}^k \\ &= \mathbf{e}_j(\mathbf{x}^k) + \left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right) \frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k} \Delta \mathbf{x}^k. \end{aligned}$$

Plugging this into the formula for  $\chi^2$ , we get:

$$\begin{aligned} \chi_{k+1}^2 &\approx \sum_{e_j \in \mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1 \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\mathbf{e}_j(\mathbf{x}^k)}_{\bullet \times 1} \\ &+ \sum_{e_j \in \mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1 \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)}_{\bullet \times dN} \underbrace{\frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k}}_{dN \times cN} \underbrace{\Delta \mathbf{x}^k}_{cN \times 1} \\ &+ \sum_{e_j \in \mathcal{E}} \underbrace{(\Delta \mathbf{x}^k)^T}_{1 \times cN} \underbrace{\left( \frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k} \right)^T}_{cN \times dN} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)^T}_{dN \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)}_{\bullet \times dN} \underbrace{\frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k}}_{dN \times cN} \underbrace{\Delta \mathbf{x}^k}_{cN \times 1} \\ &= \chi_k^2 + 2\mathbf{b}^T \Delta \mathbf{x}^k + (\Delta \mathbf{x}^k)^T H \Delta \mathbf{x}^k, \end{aligned}$$

where

$$\begin{aligned} \mathbf{b}^T &= \sum_{e_j \in \mathcal{E}} \underbrace{[\mathbf{e}_j(\mathbf{x}^k)]^T}_{1 \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)}_{\bullet \times dN} \underbrace{\frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k}}_{dN \times cN} \\ H &= \sum_{e_j \in \mathcal{E}} \underbrace{\left( \frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k} \right)^T}_{cN \times dN} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)^T}_{dN \times \bullet} \underbrace{\Omega_j}_{\bullet \times \bullet} \underbrace{\left( \frac{\partial \mathbf{e}_j(\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)} \bigg|_{\Delta \mathbf{x}^k=0} \right)}_{\bullet \times dN} \underbrace{\frac{\partial (\mathbf{x}^k \boxplus \Delta \mathbf{x}^k)}{\partial \Delta \mathbf{x}^k}}_{dN \times cN}. \end{aligned}$$

Using this notation, we obtain the optimal update as

$$\Delta \mathbf{x}^k = -H^{-1} \mathbf{b}.$$

We apply this update to the poses and repeat until convergence.

**class** graphslam.graph.Graph (*edges*, *vertices*)

Bases: object

A graph that will be optimized via Graph SLAM.

#### Parameters

- **edges** (*list* [graphslam.edge.base\_edge.BaseEdge]) – A list of the vertices in the graph

- **vertices** (*list*[*graphslam.vertex.Vertex*]) – A list of the vertices in the graph

**`_chi2`**

The current  $\chi^2$  error, or None if it has not yet been computed

**Type** float, None

**`_edges`**

A list of the edges (i.e., constraints) in the graph

**Type** list[*graphslam.edge.base\_edge.BaseEdge*]

**`_gradient`**

The gradient **b** of the  $\chi^2$  error, or None if it has not yet been computed

**Type** numpy.ndarray, None

**`_hessian`**

The Hessian matrix *H*, or None if it has not yet been computed

**Type** scipy.sparse.lil\_matrix, None

**`_vertices`**

A list of the vertices in the graph

**Type** list[*graphslam.vertex.Vertex*]

**`_calc_chi2_gradient_hessian()`**

Calculate the  $\chi^2$  error, the gradient **b**, and the Hessian *H*.

**`_link_edges()`**

Fill in the *vertices* attributes for the graph's edges.

**`calc_chi2()`**

Calculate the  $\chi^2$  error for the Graph.

**Returns** The  $\chi^2$  error

**Return type** float

**`optimize(tol=0.0001, max_iter=20, fix_first_pose=True)`**

Optimize the  $\chi^2$  error for the Graph.

**Parameters**

- **tol** (*float*) – If the relative decrease in the  $\chi^2$  error between iterations is less than *tol*, we will stop
- **max\_iter** (*int*) – The maximum number of iterations
- **fix\_first\_pose** (*bool*) – If True, we will fix the first pose

**`plot(vertex_color='r', vertex_marker='o', vertex_markersize=3, edge_color='b', title=None)`**

Plot the graph.

**Parameters**

- **vertex\_color** (*str*) – The color that will be used to plot the vertices
- **vertex\_marker** (*str*) – The marker that will be used to plot the vertices
- **vertex\_markersize** (*int*) – The size of the plotted vertices
- **edge\_color** (*str*) – The color that will be used to plot the edges
- **title** (*str, None*) – The title that will be used for the plot

**to\_g2o** (*outfile*)

Save the graph in .g2o format.

**Parameters** **outfile** (*str*) – The path where the graph will be saved

**class** graphslam.graph.\_Chi2GradientHessian (*dim*)

Bases: object

A class that is used to aggregate the  $\chi^2$  error, gradient, and Hessian.

**Parameters** **dim** (*int*) – The compact dimensionality of the poses

**chi2**

The  $\chi^2$  error

**Type** float

**dim**

The compact dimensionality of the poses

**Type** int

**gradient**

The contributions to the gradient vector

**Type** defaultdict

**hessian**

The contributions to the Hessian matrix

**Type** defaultdict

**static update** (*chi2\_grad\_hess, incoming*)

Update the  $\chi^2$  error and the gradient and Hessian dictionaries.

**Parameters**

- **chi2\_grad\_hess** (*\_Chi2GradientHessian*) – The *\_Chi2GradientHessian* that will be updated
- **incoming** (*tuple*) – TODO

## graphslam.load module

Functions for loading graphs.

graphslam.load.**load\_g2o\_se2** (*infile*)

Load an  $SE(2)$  graph from a .g2o file.

**Parameters** **infile** (*str*) – The path to the .g2o file

**Returns** The loaded graph

**Return type** *Graph*

graphslam.load.**load\_g2o\_se3** (*infile*)

Load an  $SE(3)$  graph from a .g2o file.

**Parameters** **infile** (*str*) – The path to the .g2o file

**Returns** The loaded graph

**Return type** *Graph*

## graphslam.util module

Utility functions used throughout the package.

`graphslam.util.neg_pi_to_pi` (*angle*)

Normalize *angle* to be in  $[-\pi, \pi)$ .

**Parameters** *angle* (*float*) – An angle (in radians)

**Returns** The angle normalized to  $[-\pi, \pi)$

**Return type** *float*

`graphslam.util.solve_for_edge_dimensionality` (*n*)

Solve for the dimensionality of an edge.

In a .g2o file, an edge is specified as `<estimate> <information matrix>`, where only the upper triangular portion of the matrix is provided.

This solves the problem:

$$d + \frac{d(d+1)}{2} = n$$

**Returns** The dimensionality of the edge

**Return type** *int*

`graphslam.util.upper_triangular_matrix_to_full_matrix` (*arr*, *n*)

Given an upper triangular matrix, return the full matrix.

**Parameters**

- **arr** (*np.ndarray*) – The upper triangular portion of the matrix
- **n** (*int*) – The size of the matrix

**Returns** *mat* – The full matrix

**Return type** *np.ndarray*

## graphslam.vertex module

A `Vertex` class.

**class** `graphslam.vertex.Vertex` (*vertex\_id*, *pose*, *vertex\_index=None*)

Bases: `object`

A class for representing a vertex in Graph SLAM.

**Parameters**

- **vertex\_id** (*int*) – The vertex's unique ID
- **pose** (`graphslam.pose.base_pose.BasePose`) – The pose associated with the vertex
- **vertex\_index** (*int*, *None*) – The vertex's index in the graph's vertices list

**id**

The vertex's unique ID

**Type** *int*

**index**

The vertex's index in the graph's vertices list

**Type** int, None

**pose**

The pose associated with the vertex

**Type** *graphslam.pose.base\_pose.BasePose*

**plot** (*color='r', marker='o', markersize=3*)

Plot the vertex.

**Parameters**

- **color** (*str*) – The color that will be used to plot the vertex
- **marker** (*str*) – The marker that will be used to plot the vertex
- **markersize** (*int*) – The size of the plotted vertex

**to\_g2o** ()

Export the vertex to the .g2o format.

**Returns** The vertex in .g2o format

**Return type** str

### 1.1.3 Module contents

Graph SLAM solver in Python.

Documentation for this package can be found at <https://python-graphslam.readthedocs.io/>.

This package implements a Graph SLAM solver in Python.





## FEATURES

- Optimize  $\mathbb{R}^2$ ,  $\mathbb{R}^3$ ,  $SE(2)$ , and  $SE(3)$  datasets
- Analytic Jacobians
- Supports odometry edges
- Import and export .g2o files for  $SE(2)$  and  $SE(3)$  datasets



## INSTALLATION

```
pip install graphslam
```



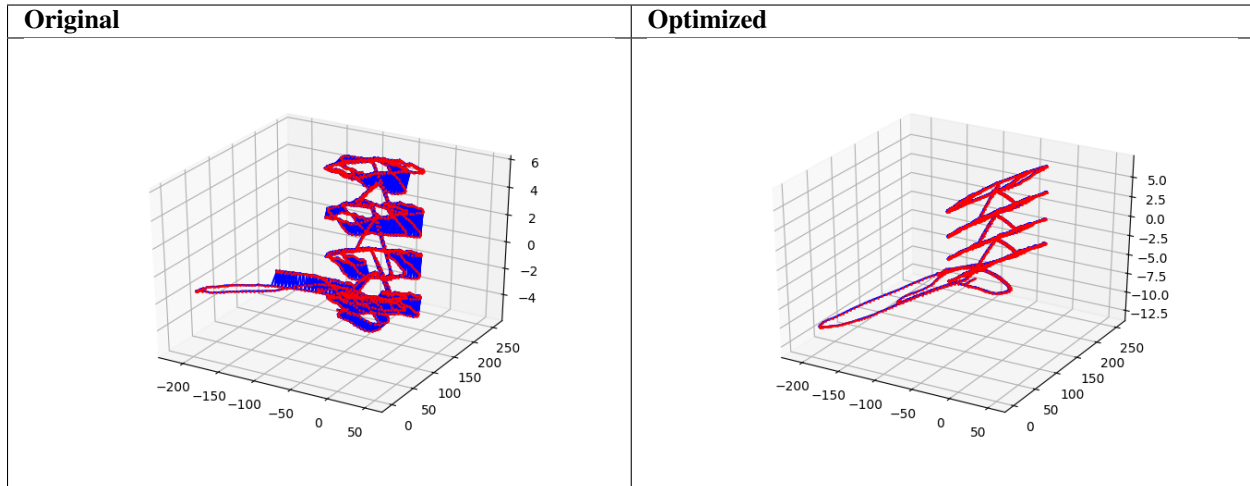
## EXAMPLE USAGE

### 4.1 SE(3) Dataset

```
>>> from graphslam.load import load_g2o_se3
>>> g = load_g2o_se3("parking-garage.g2o") # https://lucacarlone.mit.edu/datasets/
>>> g.plot(vertex_markersize=1)
>>> g.calc_chi2()
16720.020602489112
>>> g.optimize()
>>> g.plot(vertex_markersize=1)
```

**Output:**

Iteration	chi <sup>2</sup>	rel. change
0	16720.0206	
1	26.5495	-0.998412
2	1.2712	-0.952119
3	1.2402	-0.024439
4	1.2396	-0.000456
5	1.2395	-0.000091



## 4.2 SE(2) Dataset

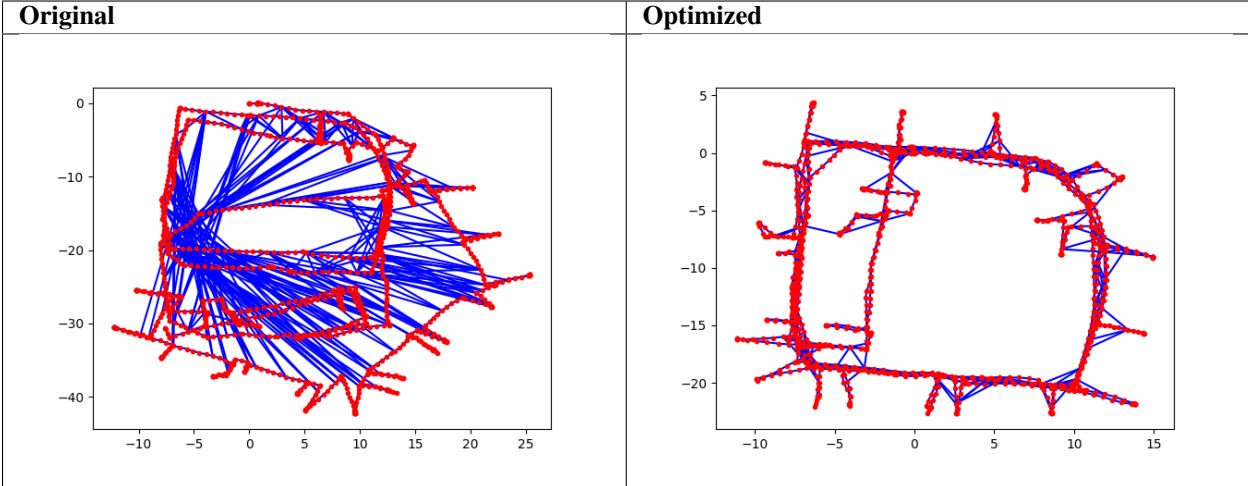
```

>>> from graphslam.load import load_g2o_se2
>>> g = load_g2o_se2("input_INTEL.g2o") # https://lucacarlone.mit.edu/datasets/
>>> g.plot()
>>> g.calc_chi2()
7191686.382493544
>>> g.optimize()
>>> g.plot()

```

### Output:

Iteration	chi <sup>2</sup>	rel. change
0	7191686.3825	
1	319915276.1284	43.484042
2	124894535.1749	-0.609601
3	338185.8171	-0.997292
4	734.5142	-0.997828
5	215.8405	-0.706145
6	215.8405	-0.000000







## REFERENCES AND LINKS

- [A tutorial on graph-based SLAM](#)
- [A tutorial on SE\(3\) transformation parameterizations and on-manifold optimization](#)
- [Datasets from Luca Carlone](#)



## LIVE CODING GRAPH SLAM IN PYTHON

If you're interested, you can watch as I coded this up.

1. [Live coding Graph SLAM in Python \(Part 1\)](#)
2. [Live coding Graph SLAM in Python \(Part 2\)](#)
3. [Live coding Graph SLAM in Python \(Part 3\)](#)
4. [Live coding Graph SLAM in Python \(Part 4\)](#)
5. [Live coding Graph SLAM in Python \(Part 5\)](#)



## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### g

- graphslam, 19
- graphslam.edge, 4
- graphslam.edge.base\_edge, 1
- graphslam.edge.edge\_odometry, 3
- graphslam.graph, 14
- graphslam.load, 17
- graphslam.pose, 14
- graphslam.pose.base\_pose, 4
- graphslam.pose.r2, 6
- graphslam.pose.r3, 8
- graphslam.pose.se2, 10
- graphslam.pose.se3, 12
- graphslam.util, 18
- graphslam.vertex, 18





Symbols

`_Chi2GradientHessian` (class in `graphslam.graph`), 17  
`_calc_chi2_gradient_hessian()` (`graphslam.graph.Graph` method), 16  
`_calc_jacobian()` (`graphslam.edge.base_edge.BaseEdge` method), 2  
`_chi2` (`graphslam.graph.Graph` attribute), 16  
`_edges` (`graphslam.graph.Graph` attribute), 16  
`_gradient` (`graphslam.graph.Graph` attribute), 16  
`_hessian` (`graphslam.graph.Graph` attribute), 16  
`_link_edges()` (`graphslam.graph.Graph` method), 16  
`_vertices` (`graphslam.graph.Graph` attribute), 16

B

`BaseEdge` (class in `graphslam.edge.base_edge`), 1  
`BasePose` (class in `graphslam.pose.base_pose`), 4

C

`calc_chi2()` (`graphslam.edge.base_edge.BaseEdge` method), 2  
`calc_chi2()` (`graphslam.graph.Graph` method), 16  
`calc_chi2_gradient_hessian()` (`graphslam.edge.base_edge.BaseEdge` method), 2  
`calc_error()` (`graphslam.edge.base_edge.BaseEdge` method), 2  
`calc_error()` (`graphslam.edge.edge_odometry.EdgeOdometry` method), 3  
`calc_jacobians()` (`graphslam.edge.base_edge.BaseEdge` method), 2  
`calc_jacobians()` (`graphslam.edge.edge_odometry.EdgeOdometry` method), 3  
`chi2` (`graphslam.graph._Chi2GradientHessian` attribute), 17  
`copy()` (`graphslam.pose.base_pose.BasePose` method), 4

`copy()` (`graphslam.pose.r2.PoseR2` method), 6  
`copy()` (`graphslam.pose.r3.PoseR3` method), 8  
`copy()` (`graphslam.pose.se2.PoseSE2` method), 10  
`copy()` (`graphslam.pose.se3.PoseSE3` method), 12

D

`dim` (`graphslam.graph._Chi2GradientHessian` attribute), 17

E

`EdgeOdometry` (class in `graphslam.edge.edge_odometry`), 3  
`EPSILON` (in module `graphslam.edge.base_edge`), 2  
`estimate` (`graphslam.edge.base_edge.BaseEdge` attribute), 1  
`estimate` (`graphslam.edge.edge_odometry.EdgeOdometry` attribute), 3

F

`from_matrix()` (`graphslam.pose.se2.PoseSE2` class method), 10

G

`gradient` (`graphslam.graph._Chi2GradientHessian` attribute), 17  
`Graph` (class in `graphslam.graph`), 15  
`graphslam` (module), 19  
`graphslam.edge` (module), 4  
`graphslam.edge.base_edge` (module), 1  
`graphslam.edge.edge_odometry` (module), 3  
`graphslam.graph` (module), 14  
`graphslam.load` (module), 17  
`graphslam.pose` (module), 14  
`graphslam.pose.base_pose` (module), 4  
`graphslam.pose.r2` (module), 6  
`graphslam.pose.r3` (module), 8  
`graphslam.pose.se2` (module), 10  
`graphslam.pose.se3` (module), 12  
`graphslam.util` (module), 18  
`graphslam.vertex` (module), 18

## H

`hessian` (*graphslam.graph.\_Chi2GradientHessian attribute*), 17

## I

`id` (*graphslam.vertex.Vertex attribute*), 18

`index` (*graphslam.vertex.Vertex attribute*), 18

`information` (*graphslam.edge.base\_edge.BaseEdge attribute*), 1

`information` (*graphslam.edge.edge\_odometry.EdgeOdometry attribute*), 3

`inverse()` (*graphslam.pose.base\_pose.BasePose property*), 4

`inverse()` (*graphslam.pose.r2.PoseR2 property*), 6

`inverse()` (*graphslam.pose.r3.PoseR3 property*), 8

`inverse()` (*graphslam.pose.se2.PoseSE2 property*), 10

`inverse()` (*graphslam.pose.se3.PoseSE3 property*), 12

## J

`jacobian_boxplus()` (*graphslam.pose.base\_pose.BasePose method*), 4

`jacobian_boxplus()` (*graphslam.pose.r2.PoseR2 method*), 6

`jacobian_boxplus()` (*graphslam.pose.r3.PoseR3 method*), 8

`jacobian_boxplus()` (*graphslam.pose.se2.PoseSE2 method*), 10

`jacobian_boxplus()` (*graphslam.pose.se3.PoseSE3 method*), 12

`jacobian_self_ominus_other_wrt_other()` (*graphslam.pose.base\_pose.BasePose method*), 4

`jacobian_self_ominus_other_wrt_other()` (*graphslam.pose.r2.PoseR2 method*), 6

`jacobian_self_ominus_other_wrt_other()` (*graphslam.pose.r3.PoseR3 method*), 8

`jacobian_self_ominus_other_wrt_other()` (*graphslam.pose.se2.PoseSE2 method*), 10

`jacobian_self_ominus_other_wrt_other()` (*graphslam.pose.se3.PoseSE3 method*), 12

`jacobian_self_ominus_other_wrt_other_compact()` (*graphslam.pose.base\_pose.BasePose method*), 4

`jacobian_self_ominus_other_wrt_other_compact()` (*graphslam.pose.r2.PoseR2 method*), 6

`jacobian_self_ominus_other_wrt_other_compact()` (*graphslam.pose.r3.PoseR3 method*), 8

`jacobian_self_ominus_other_wrt_other_compact()` (*graphslam.pose.se2.PoseSE2 method*), 10

`jacobian_self_ominus_other_wrt_other_compact()` (*graphslam.pose.se3.PoseSE3 method*), 12

`jacobian_self_ominus_other_wrt_self()` (*graphslam.pose.base\_pose.BasePose method*), 4

`jacobian_self_ominus_other_wrt_self()` (*graphslam.pose.r2.PoseR2 method*), 6

`jacobian_self_ominus_other_wrt_self()` (*graphslam.pose.r3.PoseR3 method*), 8

`jacobian_self_ominus_other_wrt_self()` (*graphslam.pose.se2.PoseSE2 method*), 10

`jacobian_self_ominus_other_wrt_self()` (*graphslam.pose.se3.PoseSE3 method*), 13

`jacobian_self_ominus_other_wrt_self_compact()` (*graphslam.pose.base\_pose.BasePose method*), 5

`jacobian_self_ominus_other_wrt_self_compact()` (*graphslam.pose.r2.PoseR2 method*), 6

`jacobian_self_ominus_other_wrt_self_compact()` (*graphslam.pose.r3.PoseR3 method*), 8

`jacobian_self_ominus_other_wrt_self_compact()` (*graphslam.pose.se2.PoseSE2 method*), 11

`jacobian_self_ominus_other_wrt_self_compact()` (*graphslam.pose.se3.PoseSE3 method*), 13

`jacobian_self_oplus_other_wrt_other()` (*graphslam.pose.base\_pose.BasePose method*), 5

`jacobian_self_oplus_other_wrt_other()` (*graphslam.pose.r2.PoseR2 method*), 7

`jacobian_self_oplus_other_wrt_other()` (*graphslam.pose.r3.PoseR3 method*), 9

`jacobian_self_oplus_other_wrt_other()` (*graphslam.pose.se2.PoseSE2 method*), 11

`jacobian_self_oplus_other_wrt_other()` (*graphslam.pose.se3.PoseSE3 method*), 13

`jacobian_self_oplus_other_wrt_other_compact()` (*graphslam.pose.base\_pose.BasePose method*), 5

`jacobian_self_oplus_other_wrt_other_compact()` (*graphslam.pose.r2.PoseR2 method*), 7

`jacobian_self_oplus_other_wrt_other_compact()` (*graphslam.pose.r3.PoseR3 method*), 9

`jacobian_self_oplus_other_wrt_other_compact()` (*graphslam.pose.se2.PoseSE2 method*), 11

`jacobian_self_oplus_other_wrt_other_compact()` (*graphslam.pose.se3.PoseSE3 method*), 13

`jacobian_self_oplus_other_wrt_self()` (*graphslam.pose.base\_pose.BasePose method*), 5

`jacobian_self_oplus_other_wrt_self()` (*graphslam.pose.r2.PoseR2 method*), 7

`jacobian_self_oplus_other_wrt_self()` (*graphslam.pose.r3.PoseR3 method*), 9

`jacobian_self_oplus_other_wrt_self()` (*graphslam.pose.se2.PoseSE2 method*), 11

`jacobian_self_oplus_other_wrt_self()`

(*graphslam.pose.se3.PoseSE3 method*), 13  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact() (*graphslam.pose.base\_pose.BasePose method*), 5  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_active\_for\_edge\_dimensionality() (*in module graphslam.util*), 18  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_array() (*graphslam.pose.base\_pose.BasePose method*), 5  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_array() (*graphslam.pose.r2.PoseR2 method*), 7  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_array() (*graphslam.pose.r3.PoseR3 method*), 9  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_array() (*graphslam.pose.se2.PoseSE2 method*), 11  
 jacobian\_self\_oplus\_other\_wrt\_self\_compact\_array() (*graphslam.pose.se3.PoseSE3 method*), 13

**L**  
 load\_g2o\_se2() (*in module graphslam.load*), 17  
 load\_g2o\_se3() (*in module graphslam.load*), 17

**N**  
 neg\_pi\_to\_pi() (*in module graphslam.util*), 18  
 normalize() (*graphslam.pose.se3.PoseSE3 method*), 13

**O**  
 optimize() (*graphslam.graph.Graph method*), 16  
 orientation() (*graphslam.pose.base\_pose.BasePose property*), 5  
 orientation() (*graphslam.pose.r2.PoseR2 property*), 7  
 orientation() (*graphslam.pose.r3.PoseR3 property*), 9  
 orientation() (*graphslam.pose.se2.PoseSE2 property*), 11  
 orientation() (*graphslam.pose.se3.PoseSE3 property*), 13

**P**  
 plot() (*graphslam.edge.base\_edge.BaseEdge method*), 2  
 plot() (*graphslam.edge.edge\_odometry.EdgeOdometry method*), 3  
 plot() (*graphslam.graph.Graph method*), 16  
 plot() (*graphslam.vertex.Vertex method*), 19  
 pose (*graphslam.vertex.Vertex attribute*), 19  
 PoseR2 (*class in graphslam.pose.r2*), 6  
 PoseR3 (*class in graphslam.pose.r3*), 8  
 PoseSE2 (*class in graphslam.pose.se2*), 10  
 PoseSE3 (*class in graphslam.pose.se3*), 12  
 position() (*graphslam.pose.base\_pose.BasePose property*), 5  
 position() (*graphslam.pose.r2.PoseR2 property*), 7  
 position() (*graphslam.pose.r3.PoseR3 property*), 9  
 position() (*graphslam.pose.se2.PoseSE2 property*), 11

**S**  
 position() (*graphslam.pose.se3.PoseSE3 property*), 13

**T**  
 to\_array() (*graphslam.pose.base\_pose.BasePose method*), 5  
 to\_array() (*graphslam.pose.r2.PoseR2 method*), 7  
 to\_array() (*graphslam.pose.r3.PoseR3 method*), 9  
 to\_array() (*graphslam.pose.se2.PoseSE2 method*), 11  
 to\_array() (*graphslam.pose.se3.PoseSE3 method*), 14  
 to\_compact() (*graphslam.pose.base\_pose.BasePose method*), 5  
 to\_compact() (*graphslam.pose.r2.PoseR2 method*), 7  
 to\_compact() (*graphslam.pose.r3.PoseR3 method*), 9  
 to\_compact() (*graphslam.pose.se2.PoseSE2 method*), 11  
 to\_compact() (*graphslam.pose.se3.PoseSE3 method*), 14  
 to\_g2o() (*graphslam.edge.base\_edge.BaseEdge method*), 2  
 to\_g2o() (*graphslam.edge.edge\_odometry.EdgeOdometry method*), 3  
 to\_g2o() (*graphslam.graph.Graph method*), 16  
 to\_g2o() (*graphslam.vertex.Vertex method*), 19  
 to\_matrix() (*graphslam.pose.se2.PoseSE2 method*), 12  
 to\_matrix() (*graphslam.pose.se3.PoseSE3 method*), 14

**U**  
 update() (*graphslam.graph.\_Chi2GradientHessian static method*), 17  
 upper\_triangular\_matrix\_to\_full\_matrix() (*in module graphslam.util*), 18

**V**  
 Vertex (*class in graphslam.vertex*), 18  
 vertex\_ids (*graphslam.edge.base\_edge.BaseEdge attribute*), 1  
 vertices (*graphslam.edge.base\_edge.BaseEdge attribute*), 1  
 vertices (*graphslam.edge.edge\_odometry.EdgeOdometry attribute*), 3