2010

SumProductLab Reference

R3.00

This manual describes the usage of each factor node in the suite

Henry Leung

1/22/2010

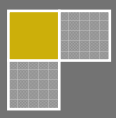


Table of Contents

1 Introduction

The SumProductLab provides a set of basic factor nodes for building up a factor graph. One can try out ideas by instantiating the necessary constraint nodes, connecting them up, and giving some evidence. The sum-product (or belief propagation) algorithm will compute the message to each node in the entire network. Finally, marginal probability of any variable in the graph can then be calculated.

Factor graphs can be used to model a wide range of systems. That means the same algorithm can be used to solve problems of different natures. For more information on factor graphs, please refer to:

An Introduction to Factor Graphs by Hans-Andrea Loeliger; IEEE Signal Processing Magazine, January 2004

2 The Forney Factor Graph, FFG

The FFG is made up of nodes and edges. Each edge represents a variable, while every factor node represents the constraints applied to the edges connected to the node. In FFG, only the constraint nodes are involved in the sum-product algorithm while the edges (variables) are not. Therefore, only factor nodes are implemented.

3 Operating modes

3.1 Non-loopy mode (default)

After the factor nodes are created, connected, and necessary evidence is supplied, the sum-product algorithm will update messages throughout the entire network without intervention from a supervisory program.

3.2 Loopy mode

When the global variable (*loopy*) is set to one, the factor nodes will work in loopy mode. In this mode, the factor nodes are created, connected, and necessary evidence is supplied but messages will not be updated until the *update_node()* method is called. Therefore, a supervisory program must be there to schedule the update of nodes. For factors graphs involve linear-scalar messages, set the global variable (*linear_scalar)* to one.

4 Message

4.1 Discrete message

Each message is implemented by a row vector in which each element represents the probability in each case, and the sum of the vector is 1. In a bi-value message, there are two elements in the vector. The first and second elements represent the probability of *False* and *True* respectively.

4.2 Gaussian continuous message

For linear and scalar graphs, the message is represented by a row vector with two elements. The first element is the mean (real or complex) and the second element is the variance (real).

4.3 Neutral message

Neutral message is loaded to an evident node when it does not carry any evidence. For discrete message, it is a vector with all 1's with magnitude normalized to unity. For Gaussian continuous message, it is zero mean and infinite variance.

5 Connection

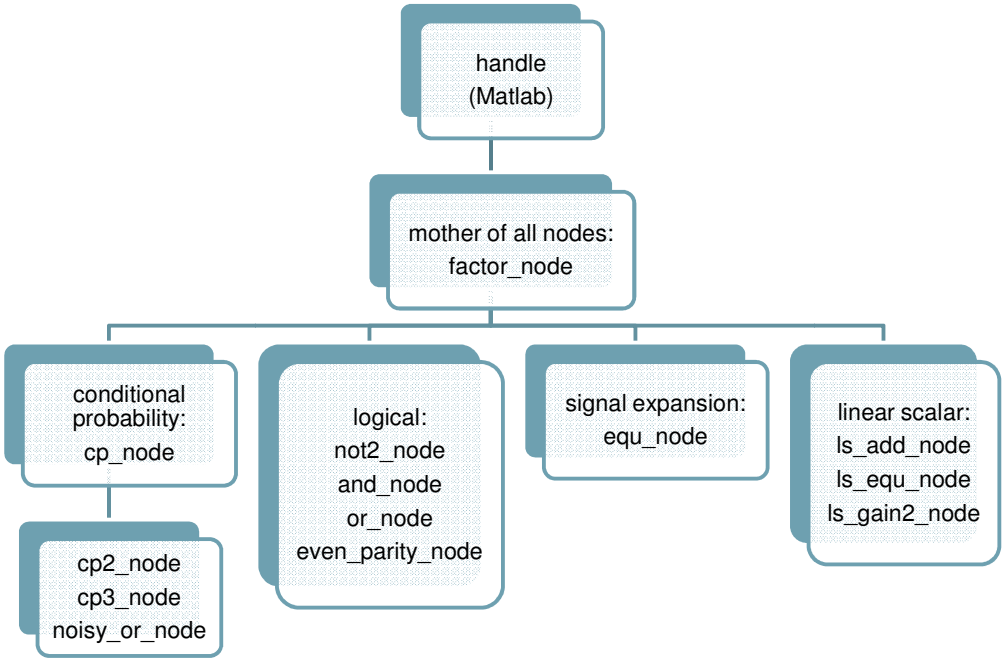
The connections of the graph are defined by the linklist in the nodes. The linklist of a node is a 1x*N* cell array specifying the other nodes connected. The order is critical for nodes with directional connection like the cp_node (conditional probability), or_node, and and_node, in which some of the connections are directed to the parents and one of the connections is directed to the child. In these cases, the convention is to put the child on the right end of the linklist.

6 Debugging

Information such as in-bound messages and out-bound messages are available in the node for debugging purpose. The in-bound and out-bound messages are stored in the inbound_msg and outbound_msg respectively. Both of them are in *N*x1 cell array. A unique integer is given as ID when instantiating a node. When an in-bound message is received, the ID of the sending node is saved in the from_id array. When an out-bound message is sent, the ID of the destination node is saved in the to_id array.

7 Hierarchy of factor nodes

The dependency of the classes in the suite is shown in the following chart:



8 Description of factor nodes

8.1 factor_node

This is the mother of all factor nodes. It provides all the necessary setup utilities as well as the sum-product algorithm. The factor_fun provides a place holder for specific constraints of derived nodes.

8.1.1 Superclass

handle class from Matlab.

8.1.2 Properties

```
id              % id of the object
linklist        % linklist, 1xn cell array
link_id         % link_id, 1xn array
state           % state of the node
inbound_msg     % inbound message, nx1 cell array
from_node       % from node, 1xn cell array
from_id         % from id, 1xn array
outbound_msg    % outbound message nx1 cell array
to_node         % to node, nx1 cell array
to_id           % to id, 1xn array
missing_node    % missing node, 1xn cell array
```

8.1.3 Methods

8.1.3.1 factor_node(id)

Constructor

8.1.3.2 *reset()*

Resets the node

8.1.3.3 *setup_link(linklist)*

Specifies connectivity from the perspective of its node. Linklist is a row of cell array of nodes.

8.1.3.4 *append_link(linklist)*

Appends the linklist to the original linklist

8.1.3.5 *prepend_link(linklist)*

Prepends the linklist to the original linklist

8.1.3.6 *update_node(default_msg)*

Used in loopy mode, calculates outbound_msg using the sum-product algorithm. The default_msg, in the form of a row vector, substitutes into any missing inbound_msg for calculation.

8.1.3.7 *rx_msg(from_node,msg)*

Used in non-loopy mode, receives a message (msg) from the from_node; performs sum-product algorithm and dispatches updated message to other node.

8.1.3.8 *factor_fun(in_msg, from_id, to_id)*

An abstract function to be defined in derived nodes. In each derived class, the factor_fun specifies the message update rules.

8.2 and_node

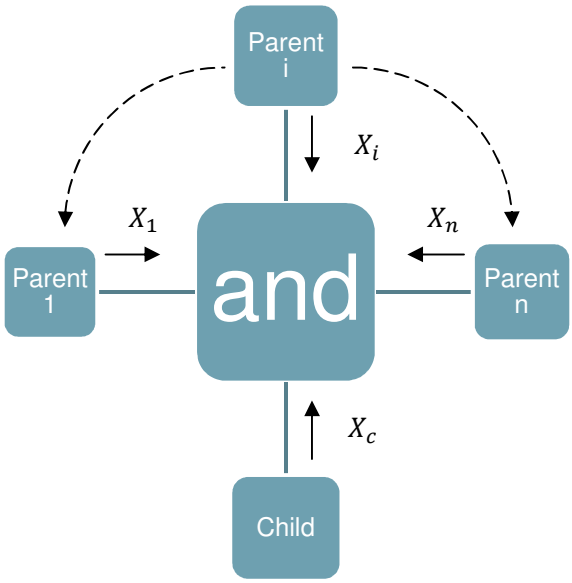
It constraints the child and parent connections to behave as an and-gate such that the child message is *False* when any of the parent messages is *False*.

The child must appear at the right end in the linklist.

8.2.1 Special features

Bi-value	Yes
Directional connection	Yes
Number of connections	≥ 3

8.2.2 Message update rule



Returns msg. The calculations are different for message to child and to parent.

8.2.2.1 *Message to the child*

$$msg(2) = \prod_i X_i(2)$$

$$msg(1) = 1 - msg(2)$$

8.2.2.2 *Message to parent k*

$$msg(2) = X_c(1) + \left[\prod_{\forall i \neq k} X_i(2)\right][X_c(2) - X_c(1)]$$

$$msg(1) = X_c(1)$$

8.3 cp_node

This is the superclass of all the conditional probability nodes. It defines the conditional probability table, cpt, property.

8.3.1 Special features

Bi-value	Not necessary
Directional connection	Yes
Number of connections	≥ 2

8.3.2 Order of linklist

Since it is a directional node, the order of linklist is critical. See *Connection*.

8.3.3 Types of cpt

8.3.3.1 Multidimensional matrix

8.3.3.1.1 Convention

The order of index in the cpt is the same order of the linklist.

8.3.3.1.2 cpt table entries

For example, there are two parents (X_1, X_2) connections and one child connection (Y). The value of the table entry is:

$$cpt(Q,R,S) = P(Y = S|X_1 = Q, X_2 = T)$$

8.3.3.2 Single row array

The cpt is implemented in a row vector in which each element refers to the conditional probability of the child with respect to each parent. The order of parents is in the same order as the linklist. This type of cpt is used in noisy_or node.

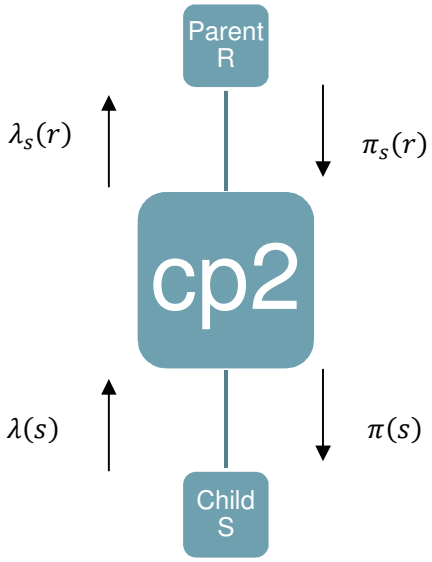
8.4 cp2_node

This node constraints the parent/child messages according to the cpt.

8.4.1 Special features

Bi-value	Not necessary
Directional connection	Yes
Number of connections	2
cpt type	Multidimensional

8.4.2 Message update rule



Let node R and S be the parent and child node respectively and let messages from parent and child be $\pi_s(r)$ and $\lambda(s)$ respectively.

8.4.2.1 *The message to parent*

$$msg = \lambda_s(r) = \sum_s \lambda(s)\, P(s|r)$$

8.4.2.2 *The message to child*

$$msg = \pi(s) = \sum_r P(s|r)\pi_s(r)$$

Where $P(s|r) = cpt(r,s)$

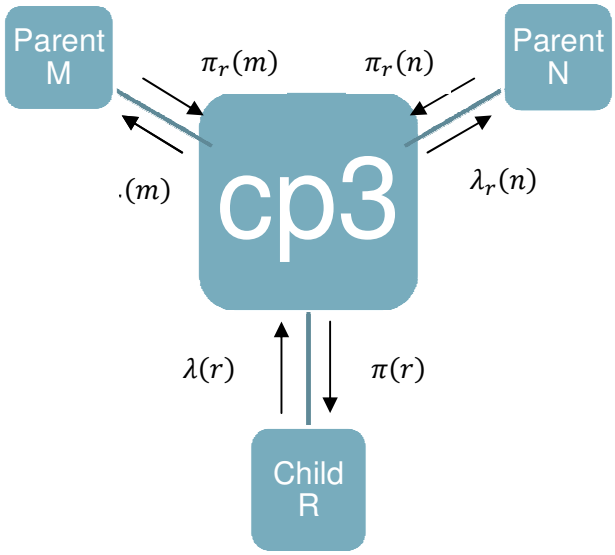
8.5 cp3_node

This node constraints the parents/child messages according to the cpt.

8.5.1 Special features

Bi-value	Not necessary
Directional connection	Yes
Number of connections	3 (2 parents, 1 child)
cpt type	Multidimensional

8.5.2 Message update rule



Let node M, N, R be the parent, parent, child node respectively.

Let messages from parent M and N be $\pi_r(m)$ and $\pi_r(n)$ respectively and message from child R be $\lambda(r)$.

8.5.2.1 Message to parent

The message to parent, M is:

$$msg = \lambda_r(m) = \sum_r \lambda(r) \sum_n P(r|m,n)\,\pi_r(n)$$

The message to parent, N is:

$$msg = \lambda_r(n) = \sum_r \lambda(r) \sum_m P(r|m,n)\,\pi_r(m)$$

8.5.2.2 Message to child

The message to child R is:

$$msg = \pi(r) = \sum_{m,n} P(r|m,n)\,\pi_r(n)\pi_r(m)$$

Where $P(r|m,n) = cpt(m,n,r)$ or $cpt(n,m,r)$ depending on the linklist setup.

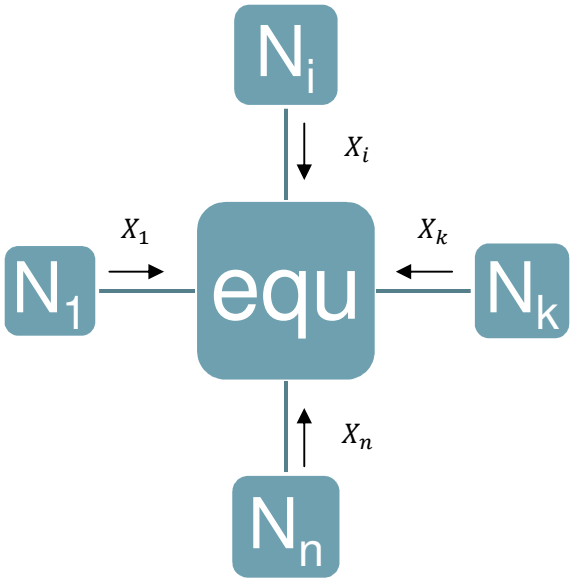
8.6 equ_node

It constraints the message from all the connections to be equal.

8.6.1 Special features

Bi-value	Not necessary
Directional connection	No
Number of connections	≥ 2

8.6.2 Message update rule



Returns message (msg) to N_k, as an elementwise product of all X_i:

$$msg = \prod_{i \neq k} X_i$$

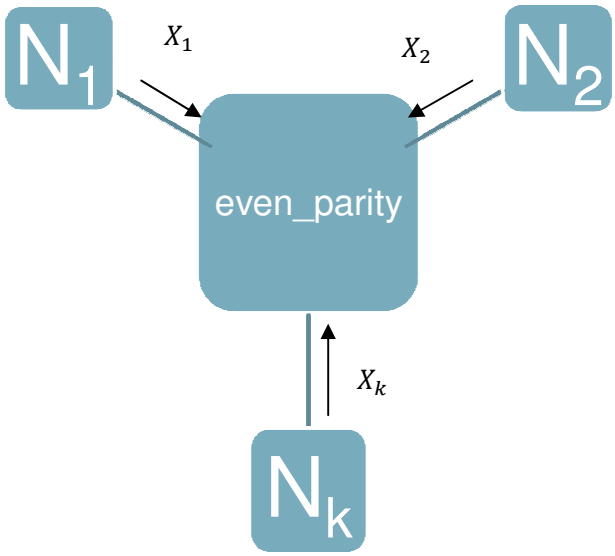
8.7 even_parity_node

It constraints the message from all the connections to be even-parity checked.

8.7.1 Special features

Bi-value	Yes
Directional connection	No
Number of connections	≥ 2

8.7.2 Message update rule



Returns msg such that:

First set

$$msg = X_1$$

To find message to N_k, apply the following operation for all X_i, where $i \neq k$

$$msg(1) = msg(1)X_i(1) + msg(2)X_i(2)$$

$$msg(2) = msg(1)X_i(2) + msg(2)X_i(1)$$

8.8 evident_node

Provides a message of observed event to a connected node.

8.8.1 Special features

Bi-value	Not necessary
Directional connection	No
Number of connections	1

8.8.2 Methods

8.8.2.1 evident_node(id)

Constructor

8.8.2.2 setup_init_msg(msg)

Dispatches msg to the connected node.

8.8.2.3 setup_link(linklist)

Setups linklist from input.

8.8.2.4 rx_msg(from_node,msg).

Stores inbound_msg and from_id

8.8.2.5 factor_fun(in_msg, from_id, to_id)

Return msg:

$$msg = in\_msg\{1\}$$

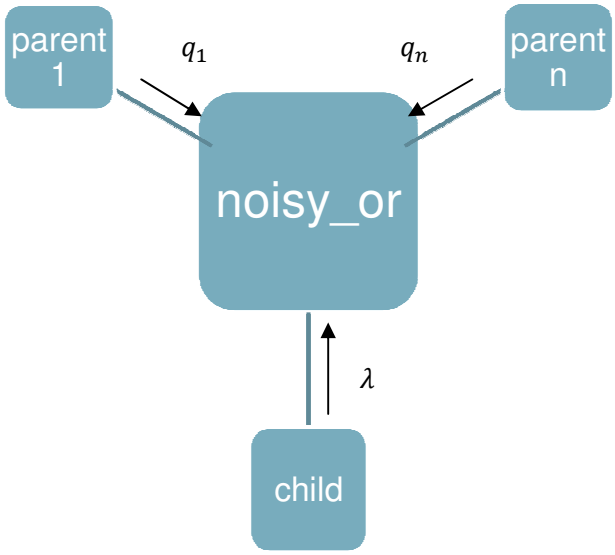
8.9 noisy_or_node

It constraints the parents/child in disjunctive interaction.

8.9.1 Special features

Bi-value	Yes
Directional connection	Yes
Number of connections	≥ 3
cpt type	Single row array

8.9.2 Message update rule



Let q_i be the probability of *True* in the message from parent i.

8.9.2.1 Message to child

$$msg(1) = \prod_i (1 - cpt(i)q_i)$$

$$msg(2) = 1 - msg(1)$$

8.9.2.2 Message to parent k

Let λ be message from child.

$$tmp = \prod_{i \neq k} (1 - cpt(i)q_i)$$

$$msg(1) = \lambda(2) - (\lambda(2) - \lambda(1))tmp$$

$$msg(2) = \lambda(2) - cpt(k)(\lambda(2) - \lambda(1))tmp$$

8.10 not2_node

It constraints the connections to be inversions of each other.

8.10.1 Special features

Bi-value	Yes
Directional connection	No
Number of connections	2

8.10.2 Message update rule

Return msg such that:

$$msg(1) = X_1(2)$$

$$msg(2) = X_1(1)$$

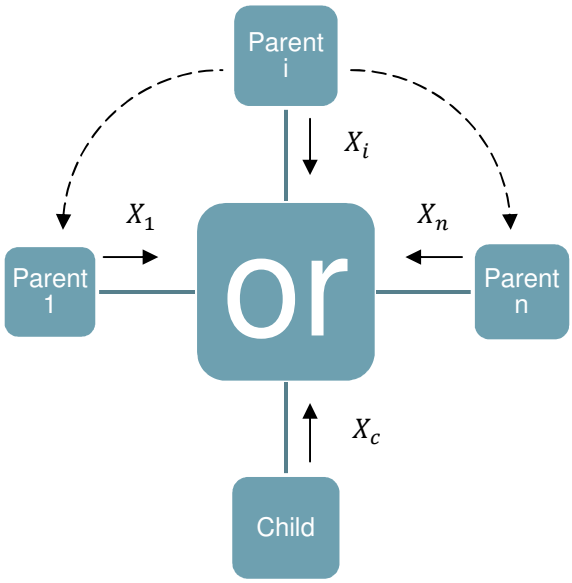
8.11 or_node

It constraints the parents/child connections to behave as an or-gate such that the child is *True* when any parent is *True*.

8.11.1 Special features

Bi-value	Yes
Directional connection	Yes
Number of connections	≥ 3

8.11.2 Message update rule



Returns msg. The calculations are different for message to child and messages to parents.

8.11.2.1 Message to the child

$$msg(1) = \prod_i X_i(1)$$

$$msg(2) = 1 - msg(1)$$

8.11.2.2 Message to parent k

$$msg(1) = X_c(2) + \left[\prod_{\forall i \neq k} X_i(1)\right][X_c(1) - X_c(2)]$$

$$msg(2) = X_c(2)$$

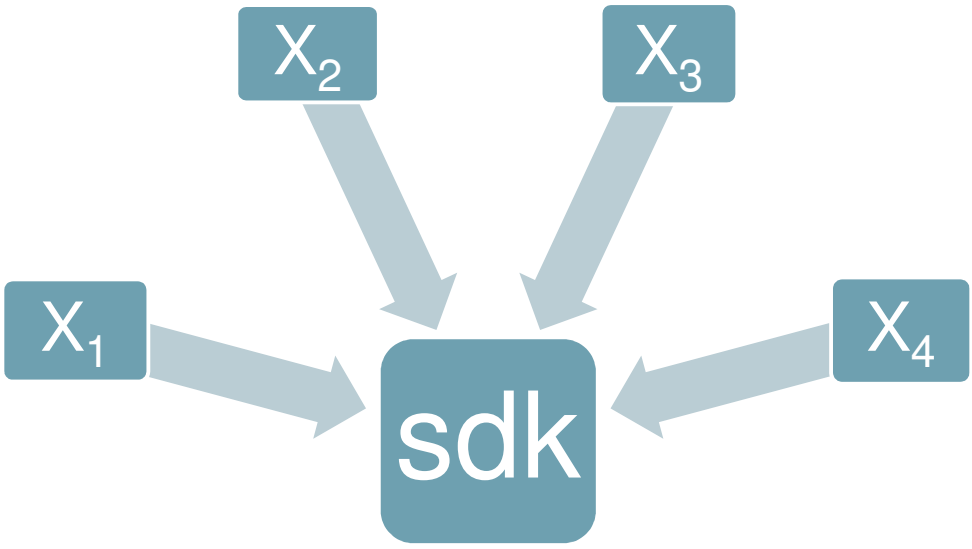
8.12 sdk_node

It is created for solving Sudoku problems with singleton. The valid configurations constrained by the sdk_node are all permutations of numbers 1 to 4 or 9 (for 4x4 or 9x9 Sudoku).

8.12.1 Special features

Bi-value	No
Directional connection	No
Number of connections	4 for 4x4 Sudoku 9 for 9x9 Sudoku

8.12.2 Message update rule for 4x4 Sudoku



Message to node 4:

$$msg(1) = \sum_{(p,q,r)\in perm(2,3,4)} X_1(p)X_2(q)X_3(r)$$

$$msg(2) = \sum_{(p,q,r)\in perm(1,3,4)} X_1(p)X_2(q)X_3(r)$$

$$msg(3) = \sum_{(p,q,r)\in perm(1,2,4)} X_1(p)X_2(q)X_3(r)$$

$$msg(4) = \sum_{(p,q,r)\in perm(1,2,3)} X_1(p)X_2(q)X_3(r)$$

There are $3!$ terms in each equation and 4 values in a message.

8.12.3 Message update rule for 9x9 Sudoku

The message update rule is the same as 4x4 Sudoku except there are more terms to sum ($8!$ terms) and more values in a message (9 values). Therefore, it requires a lot of effort to reduce the number of permutations in the terms in order to improve the speed of the algorithm.

8.13 ls_add_node

This node constraints that the child connection is the sum (or difference) of the two parent connections.

8.13.1 Special features

Linear	Yes
Message	Scalar
Directional connection	Yes
Number of connections	≥ 3
Distribution	Any distribution

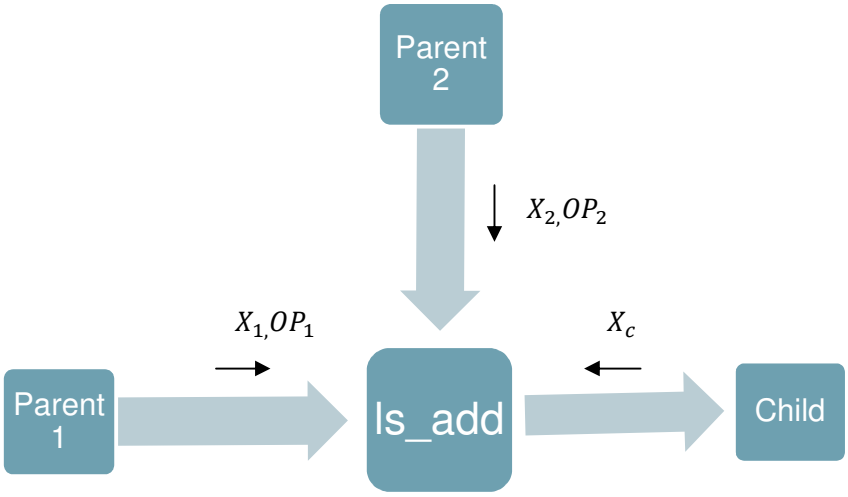
8.13.2 Property

operation - a 1xn array with values specifying the operation corresponding to each input. A '1' means addition and a '-1' means subtraction. The order must be the same as the linklist. For example, [1 -1] specified that the output mean is calculated by subtracting the mean of the second input from the mean of the first input.

8.13.3 Method setup_operation(operation_array)

This is the function for setting the operation array.

8.13.4 Message update rule



Message to Child:

$$mean = X_1(1)OP_1 + X_2(1)OP_2$$

$$variance = X_1(2) + X_2(2)$$

Message to Parent1:

$$mean = OP_1(X_c(1) - OP_2X_2(1))$$

$$variance = X_c(2) + X_2(2)$$

Where OP_1 and OP_2 are the first and second element in the operation array.

In both cases:

$$msg = [means\ variance]$$

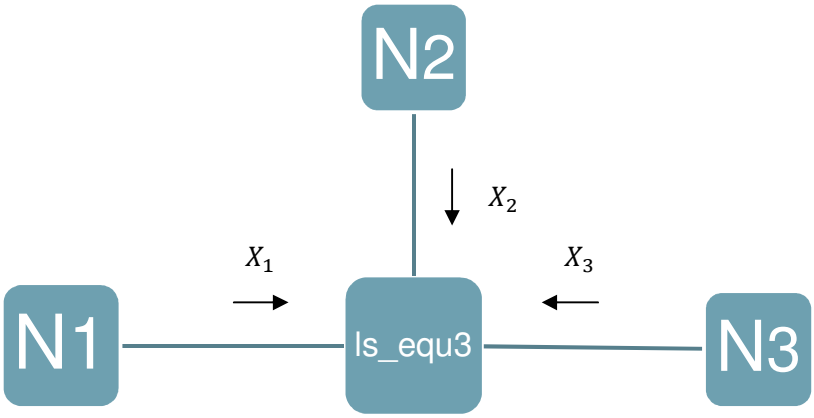
8.14 ls_equ_node

This node constraints that the child connection is the sum of the two parent connections.

8.14.1 Special features

Linear	Yes
Message	Scalar
Directional connection	No
Number of connections	≥ 2
Distribution	Gaussian

8.14.2 Message update rule



Message to N3:

$$variance = \frac{X_1(2)X_2(2)}{X_1(2) + X_2(2)}$$

$$mean = \frac{X_1(1)}{X_2(2)} + \frac{X_2(1)}{X_1(2)}$$

$$msg = [means\ variance]$$

8.15 ls_gain2_node

This node constraints the child connection is the product of the gain and the parent connection.

8.16 Special features

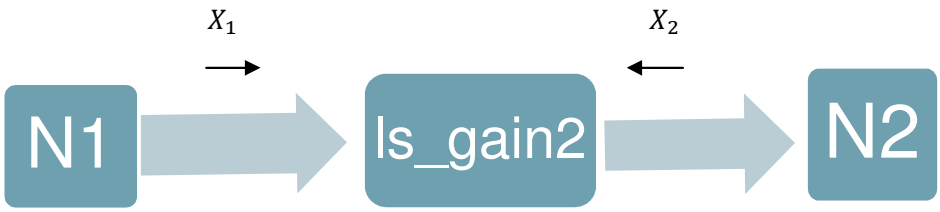
Linear	Yes
Message	Scalar
Directional connection	Yes
Number of connections	2
Distribution	Gaussian

8.17 Method

8.17.1 setup_gain(gain)

Setup the gain factor property.

8.18 Message update rule



Message to child N2:

$$variance = gain^2 \, X_1(2)$$

$$mean = gain \, X_1(1)$$

Message to parent N1:

$$variance = X_2(2)/gain^2$$

$$mean = X_2(1)/gain$$

In both cases:

$$msg = [means\ variance]$$

9 Support functions

9.1 connection(M,N)

This function connects factor node M to factor node N such that N is append to linklist of M and M is prepend to linklist of N.

9.2 marginal(M,N)

This function calculates the marginal probability of an edge (variable) between two factor nodes (M & N) after the whole network is updated.

9.3 ls_marginal(M,N)

This function calculates the marginal probability of an edge (variable) between two linear/scalar factor nodes (M & N) after the whole network is updated.

10 Design flows

10.1 Factor graph without cycle

- Instantiate nodes with unique integer for each node id
- Define connections by *setup_link()* or *connect()* functions
- Initialize parameters like *cpt* for cpt_node and *gain* for ls_gain2_node
- Give evident by *setup_init_msg()* function
- Get marginal values by *marginal()* or *ls_marginal()* functions

10.2 Factor graph with cycles

- Initialize global variable *loopy* and set it to 1.
- Initialize *linear_scalar* to 1 for linear scalar graphs
- Instantiate nodes with unique integer for each node id
- Define connections by *setup_link()* or connect() functions
- Initialize parameters if any
- Give evident by *setup_init_msg()* function
- Update each node by *update_node()* function in an iterative update schedule
- Terminate update at termination condition or after a pre-defined iterations

11 Create your own factor node

You can create your own custom factor node in the following procedures:

11.1 Modify the template.m

Change the file name and the words “template” in the file to the name of the node.

11.2 Add code to factor_fun

Add code to factor_fun that constraints the valid configurations of all the messages. Below is an example to create a 3-port even-parity node:

11.2.1 The truth table of a 3-port even-parity node

X_1	X_2	X_3
0	0	0
0	1	1
1	0	1
1	1	0

11.2.2 Code return message from incoming messages

Now assume incoming messages $in\_msg\{1\}$ and $in\_msg\{2\}$ come from X_1 and X_2 and return message (msg) goes to X_3.

X_3 is zero when both X_1 and X_2 are the same. Then

$$X_3(0) = X_1(0)X_2(0) + X_1(1)X_2(1)$$

$$msg(1) = in\_msg\{1\}(1) \times in\_msg\{2\}(1) + in\_msg\{1\}(2) \times in\_msg\{2\}(2)$$

X_3 is one when X_1 and X_2 are opposite. Then

$$X_3(1) = X_1(0)X_2(1) + X_1(1)X_2(0)$$

$$msg(2) = in\_msg\{1\}(1) \times in\_msg\{2\}(2) + in\_msg\{1\}(2) \times in\_msg\{2\}(1)$$

11.2.3 Determine the order of messages for directional nodes

In case of directional nodes, the in-bound messages, the linklist, and the index of the cpt table (if applicable) must be correctly associated before any calculation.

11.2.4 Simplify the calculation

The speed of calculation is heavily affected by the number of product terms involved. In turn, the number of product terms is determined by the number of values in the message and the number of in-bound messages. It is critically important to reduce the number of product terms to make the algorithm practical for an application.