PGMs session 7 Representation: Gaussian factors

So far in this module, we have exclusively looked at PGMs with discrete random variables. We now move on to PGMs with continuous random variables, and specifically Gaussian random variables with linear relationships between them. This assignment focuses on two representations of a Gaussian factor – the moment (or mean-covariance) form and the canonical form – as well as factor operations.

1 Preparation: Read Koller & Friedman Chapter 7 and Section 14.2 on Gaussian networks

Notes:

- In Chapter 7, note the following:
  - The moment representation of the multivariate Gaussian factor and the interpretation of the mean vector and covariance matrix (Subsection 7.1.1)
  - Marginalisation over discrete random variables is defined as a *summation*; marginalisation over continuous random variables is defined as *integration*. The definition of the rest of the operations are the same between discrete and continuous random variables (Subsection 7.1.2).
  - Make sure you understand how to read off independencies and conditional independencies between random variables from the covariance matrix and information matrix (Subsection 7.1.3).
  - In Sections 7.2 and 7.3, make sure you understand the concept of a linear Gaussian; the rest is not that important.
- In Section 14.2, note the following:
  - Make sure you are familiar with the canonical representation and the 4 operations (multiplication, division, marginalisation, reduction) for the canonical form (Subsection 14.2.1).
  - Understand that the sum-product/belief propagation algorithm is exactly the same as for models with discrete random variables (if you define marginalisation as integration, not as summation as for the discrete case). If you understand this, then Subsections 14.2.2 and 14.2.3 are not very important.
- Barber covers some of the same ground in Section 8.4, with some slight differences in notation (specifically the order of parameters and definition of the scaling constant in the canonical representation). We will use Koller and Friedman's representation.

2 Information: Implementation of Gaussian factors in `emdw`

Multivariate Gaussian factors in `emdw` is implemented in the `SqrtMVG` class. To use Gaussian factors in `emdw` for today's assignment, you have to use the code discussed below.

Make sure you include these header files – `prlite_genmat.hpp` is necessary for vectors and matrices, and the Gaussian factor is found in `sqrtmvg.hpp`:

```
#include "prlite_genmat.hpp"
#include "sqrtmvg.hpp"
```

The rest of the example code uses the following shorthand:

```
typedef SqrtMVG SG;
```

We will first look at the creation of a Gaussian factor in the moment representation. Suppose we have a two-dimensional Gaussian distribution with the mean vector

$$\boldsymbol{\mu} = \left[\begin{array}{c} 1 \\ 0 \end{array} \right]. \tag{1}$$

This vector is declared using

```
prlite::ColVector<double> mn(2);
```

and the element values assigned using

```
mn[0] = 1; mn0[1] = 0;
```

Also suppose the covariance matrix is the identity matrix, or

$$\boldsymbol{\Sigma} = \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array} \right]. \tag{2}$$

This matrix is created using

```
prlite::RowMatrix<double> cov(2,2);

cov(0,1) = cov(1,0) = 0.0; cov(0,0) = cov(1,1) = 1.0;
```

The Gaussian distribution, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, can now be created by

```
rcptr<Factor> ptr01(uniqptr<SG>(new SG({0,1}, mn, cov)));
```

where the identities of the random variables are 0 and 1 respectively.

You can also create the Gaussian factor directly in the canonical representation, $\mathcal{C}(\mathbf{K}, \mathbf{h}, g)$. For numerical stability, the `SqrtMVG` factor does not store the information matrix, $\mathbf{K}$, but rather its square root, $\mathbf{L}$, where $\mathbf{K} = \mathbf{L}\mathbf{L}^T$ and $\mathbf{L}$ is a lower triangular matrix. This is the so-called Cholesky decomposition of the information matrix. If you have defined the square root of the information matrix `L`, the information vector `h` and constant `g`, you can create the Gaussian factor in `emdw` as follows:

```
rcptr<Factor> ptr01(uniqptr<SG>(new SG({0,1}, L, h, g)));
```

There are some member functions of the `sqrtMVG` class that are not defined in the abstract `Factor` class. To gain access to these member functions when you refer to the Gaussian factor with a pointer to an abstract `Factor` class, you have to downcast the abstract pointer first, which you can do as follows:

```
rcptr<SG> dwnPtr; dwnPtr = dynamic_pointer_cast<SG>(ptr01);
```

You can then access member functions `getMean` (to get the mean vector), `getCov` (to get the covariance matrix), `getK` (to get the information matrix), and `getH` (to get the information vector) as follows:

```
cout << "Mean vector: " << dwnPtr->getMean();
cout << "Covariance matrix: " << dwnPtr->getCov();
cout << "Information matrix: " << dwnPtr->getK();
cout << "Information vector: " << dwnPtr->getH();
```

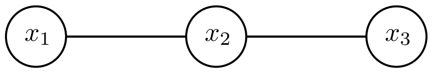
Another useful member function is `export2DMesh`, which writes a 2-D grid of values to a file; this can then be used for visualisation:

```
dwnPtr->export2DMesh("post_mesh.txt", 0, 1, 128);
```

The first argument specifies the filename, the second and third arguments specify the IDs of the random variables involved, and the fourth argument specifies the number of points per dimension. See `emdw/src/emdw-factors/sqrtmvg.hpp` for more information about the arguments and outputs, as well as for more member functions.

3 Practical: Gaussian factors and operators

Consider the following simple MRF, where x_1, x_2 and x_3 are Gaussian random variables:



The factors of this model are given by

$$\phi_1(x_1,x_2) = \mathcal{N}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}\right) \tag{3}$$

and

$$\phi_2(x_2,x_3) = \mathcal{N}\left(\begin{bmatrix} x_2 \\ x_3 \end{bmatrix}; \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \begin{bmatrix} 9 & 1 \\ 1 & 4 \end{bmatrix}\right). \tag{4}$$

(a) (Paper:) Convert these factors to the canonical form and calculate the joint distribution, $p(x_1, x_2, x_3)$, in the canonical form (you do not have to calculate the normalisation constant).

(b) (Code:) Implement these factors in `emdw` and use `emdw`'s factor operators to obtain the joint distribution, $p(x_1, x_2, x_3)$. Extract the information matrix and information vector of the joint distribution and compare it with (a). Also extract the covariance matrix of the joint distribution. Are there any variables that are statistically independent? Are there any variables that are conditionally independent? Can you come to the same conclusions by looking at the MRF structure?

(c) (Paper:) Calculate the marginal distribution $p(x_1, x_3)$ (you can again ignore the normalisation constant).

(d) (Code:) Use `emdw` to calculate the marginal distribution $p(x_1, x_3)$ and compare the result with that of (c). Also extract the information matrix and covariance matrix – what can you conclude about the (in-)dependence between x_1 and x_3? Now export a 2-D mesh of this distribution and plot it (using e.g. Python's `matplotlib`) – what is the relationship between the shape of the distribution and the (in-)dependence of the random variables?

(e) (Paper:) Calculate the conditional distribution $p(x_1, x_3|x_2 = 1)$ (you can again ignore the normalisation constant).

(f) (Code:) Use `emdw` to calculate the conditional distribution $p(x_1, x_3|x_2 = 1)$ and compare the result with that of (e). Also extract the information matrix and covariance matrix – what can you conclude about the conditional (in-)dependence between x_1 and x_3 given x_2? Now export a 2-D mesh of this distribution and plot it – what is the relationship between the shape of the distribution and the conditional (in-)dependence of the random variables?

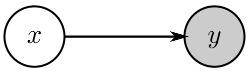
(g) (Code:) Convert the MRF to a cluster graph by placing each factor in its own cluster. By only using `emdw`'s factor operators, implement belief propagation and extract the posterior belief $p(x_2, x_3|x_1 = 0)$ after message passing.

(h) (Code:) Use `emdw`'s built-in functionality to construct the cluster graph, perform message passing, and answer the query $p(x_2, x_3|x_1 = 0)$. Compare the results with that of (g).

Something to think about: In this question, we have performed inference for a very simple Gaussian network. However, can you see how one would perform inference for a larger Gaussian network (using `emdw` built-in functionality)?

4 Practical: Inference for a very simple Bayes network

In this question, we are going to look at a very common structure that pops up in Gaussian networks – a continuous latent state that is observed by a noisy sensor (e.g. a noisy measurement of a position, voltage, temperature, liquid flow rate, etc.). This scenario is modelled by the following Bayes network, where random variable x represents the latent state and random variable y represents the noisy measurement.



The relationship between x and y is given by

$$y = x + v, \tag{5}$$

where v is zero-mean Gaussian noise, $v \sim \mathcal{N}(0, \sigma_v^2)$, that is added to the latent state to form the measurement. We also have some prior knowledge about x, which is represented by the prior distribution

$$p(x) = \mathcal{N}\left(\mu_x, \sigma_x^2\right). \tag{6}$$

For this question, use $\sigma_v = 1$, $\mu_x = 0$ and $\sigma_x = 2$. We are only going to use hand calculations here; we will look at how to implement this structure in `emdw` later in this module.

(a) From Equation 5 and the description of the noise, calculate the mean $\mu_{y|x}$ and variance $\sigma^2_{y|x}$ for the conditional distribution $p(y|x) = \mathcal{N}\left(y; \mu_{y|x}, \sigma^2_{y|x}\right)$. ($\mu_{y|x}$ is a function of x and $\sigma^2_{y|x}$ is a function of σ_v^2 – can you see it?)

(b) Calculate $p(x, y)$ and insert the evidence $y = 1$ to obtain $p(x, y = 1)$. Do this by first using the definition of the univariate Gaussian density function to write both $p(x)$ and $p(y|x)$ in exponential form, and then combining the exponents and setting $y = 1$. Now manipulate the resulting exponential form into the canonical form in terms of x, or $p(x, y = 1) = \mathcal{C}(x; k, h, g)$.

(c) Calculate the posterior distribution $p(x|y = 1)$ by normalising $p(x, y = 1)$, and convert the canonical form to the moment form. Plot both $p(x)$ and $p(x|y = 1)$ together (using e.g. Python's `matplotlib`). Can you make sense of it?

Note: The inference we have performed in this question was essentially just an application of Bayes' rule,

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}. \tag{7}$$

The different distributions in Bayes' rule have specific names, and they will pop up repeatedly in the remaining part of the module: $p(x)$ is called the *prior distribution*, $p(y|x)$ is called the *likelihood function*, and $p(x|y)$ is called the *posterior distribution*.