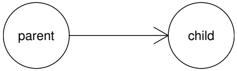
Graphical models and automating Gibbs sampling

Simon Wood, University of Edinburgh, U.K.

Automatic Gibbs sampling

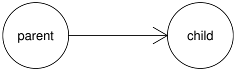
- Much of the pain can be removed from Gibbs sampling by automation.
- What makes that possible is the recognition that many Bayesian models can be abstractly represented as *Directed Acyclic Graphs*.
- Mathematically a graph* is a set of nodes connected in some way by edges. In our case
  - Nodes are variables or constants, such as data, model parameters and fixed parameters of priors.
  - Edges show dependencies between nodes.
- In a *directed* graph edges have direction, shown by arrows, with the node at the pointy end being the *child* of the *parent* at the other end. Parents directly control children.



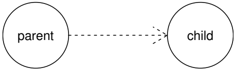
*not to be confused with a *plot*

Types of node and edge

- Nodes and edges are of different types.
- A node with no parents is a *constant*
- A continuous arrow denotes a *stochastic dependence* – the distribution of the child node depends on the parent node. In this case the child node is a *stochastic* node. e.g.



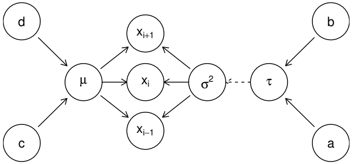
  …constant parent, stochastic child.
- A dashed arrow denotes a *deterministic dependence* – the child depends deterministically (not randomly) on the parent, and is a *deterministic* node. e.g.



  …the child is a deterministic node.

The simple example again

- Recall the model $x_i \sim N(\mu, \sigma^2)$, with priors
  - $\tau = 1/\sigma^2 \sim \text{gamma}(a, b)$, i.e. prior $\pi(\tau) = b^a \tau^{a-1} e^{-b\tau}/\Gamma(a)$
  - Independently, $\mu \sim N(c, d)$.
- Its graphical representation, showing just 3 of the x_i nodes, is



- The graph is a directed *acyclic* graph (DAG), because, following the arrows, no path ever revisits a node.
- That the x_i are independent draws from $N(\mu, \sigma^2)$ is encoded in the lack of direct edges between them.

Why the DAG is useful…

- Let z_i denote the variable corresponding to the i^{th} node of the graph. Given the DAG structure of the model, we can write the joint density over all non-constant nodes as

$$\pi(\mathbf{z}) = \prod_i \pi(z_i|\text{parent}\{z_i\})$$

- Then, from the definition of a conditional p.d.f.

$$\pi(z_j|\mathbf{z}_{-j}) = \frac{\pi(\mathbf{z})}{\int \pi(\mathbf{z})dz_j} = \frac{\prod_i \pi(z_i|\text{parent}\{z_i\})}{\int \prod_i \pi(z_i|\text{parent}\{z_i\})dz_j},$$

- But only z_j dependent terms stay in the integral, the rest cancel

$$\begin{aligned} \pi(z_j|\mathbf{z}_{-j}) &= \frac{\pi(z_j|\text{parent}\{z_j\}) \prod_{i\in\text{child}\{\text{j}\}} \pi(z_i|\text{parent}\{z_i\})}{\int \pi(z_j|\text{parent}\{z_j\}) \prod_{i\in\text{child}\{\text{j}\}} \pi(z_i|\text{parent}\{z_i\})dz_j} \\ &\propto \pi(z_j|\text{parent}\{z_j\}) \prod_{i\in\text{child}\{\text{j}\}} \pi(z_i|\text{parent}\{z_i\}), \end{aligned}$$

. . . why the DAG is useful

- So however complicated the model, and however large its DAG, the conditional density of node z_j depends only on its immediate 'family' — its parents, children, and 'partners' (other parents of its children).
- The Gibbs update of z_j only needs to know the state of those 'family' nodes.
- The fact that the DAG allows the conditionals to be modularised in this way:
    1. makes computation efficient as the conditionals only require evaluations over a small portion of the graph.
    2. facilitates automation of the process of identifying conditionals (using known results on *conjugacy* of distributions).
- JAGS (Just Another Gibbs Sampler) is a stand alone software package for automatic Gibbs sampling. `rjags` interfaces to R.

JAGS basic use

- JAGS requires that your model is specified in the JAGS language written down in a text file.
- This text file should be in the working directory, which can be checked by `getwd` and set by `setwd`.
- From within R the file is then compiled into a Gibbs sampler via a call to the function `jags.model`.
- The returned sampler object can then be used to simulate from the model posterior using the `jags.samples` function.
- An alternative to `jags.samples` is `coda.samples` which returns an object of class `mcmc.list` suitable for direct use with the `coda` package for checking and post-processing MCMC output.
- Model comparison via the *deviance information criterion* (DIC) is facilitated by `dic.samples` (requires the model to have been compiled by `jags.models` with at least `n.chains=2`).

Useful `coda` functions

- `plot.mcmc.list` plot method for MCMC simulation output.
- `acfplot` plots chain ACFs compactly.
- `autocorr` and `crosscorr` for examining within chain correlation.
- `effectiveSize` to get the effective sample sizes of simulation output.
- `HPDinterval` for highest posterior density intervals.