BUGS modeling

- The basic idea is always the same:
  - **Firstly:** define the (full) likelihood. This is the conditional probability model for all observed data
    - For example: all results from coin tossing modelled as Bernoulli-variables

      for(i in 1:n){ B_i ~ dbern(theta) }
    - If sufficient statistics exists, then you can summarize all data with that, for example:
      X ~ dbin(theta,n)
    - Written with sufficient statistics or with individual data points, the likelihood function for the unknown parameter theta is the same. Only this matters!

BUGS modeling

- The basic idea is always the same:
  - **Secondly:** define the (full) prior for all parameters (if there are many)
    - For example: theta ~ dbeta(1,1)
    - or for each parameter independently:
      for(i in 1:k){ theta[i] ~ dbeta(…) }
    - or jointly as a multivariate distribution, e.g. theta[1:k] ~ ddirich(a[1:k])
    - or hierarchically, e.g.
    - theta[1] ~ dunif(0,1); theta[2] <- dunif(0,theta[1])
    - Even in case of prior independence, parameters can still be dependent in the resulting posterior.
    - The same prior can be constructed in different ways too. For example these are equivalent for theta:

```
theta ~ dbeta(1,1)
theta ~ dunif(0,1)
theta <- phi(a);   a ~ dnorm(0,1)
```

BUGS modeling

- The basic idea is always the same:
  - **Thirdly:** it helps to figure out the model as a DAG first
    - This helps to keep track of what is in the model, and that each part is defined:
      - All observed data values ”X” have a conditional probability model, which depends on parameters ”theta”. In a DAG this is drawn as ”theta → X” for each X.
      - All parameters have assigned priors, or further distributions, which depend on further parameters.
      - No cycles in model definition! It should constitute Acyclic Graph!
      - Keep in mind that you are constructing the logical definition of the elements needed in Bayes theorem: the prior and the likelihood.

BUGS modeling

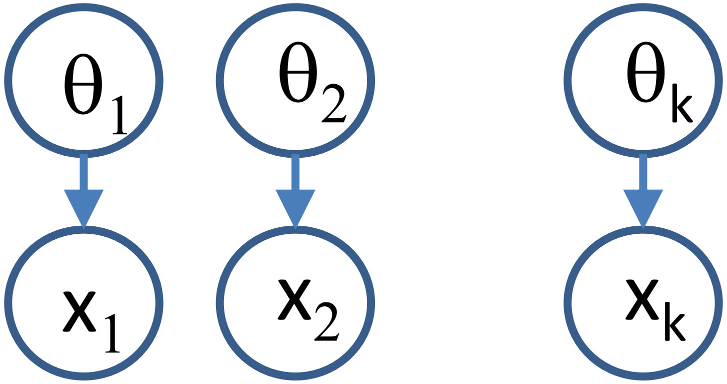
- Collect several definitions by using indexing and looping:
  - For example: separate models for each X:

    for(i in 1:K){ X[i] ~ dbin(theta[i] , N[i]) }
  - For example: models with common parameter for each X:

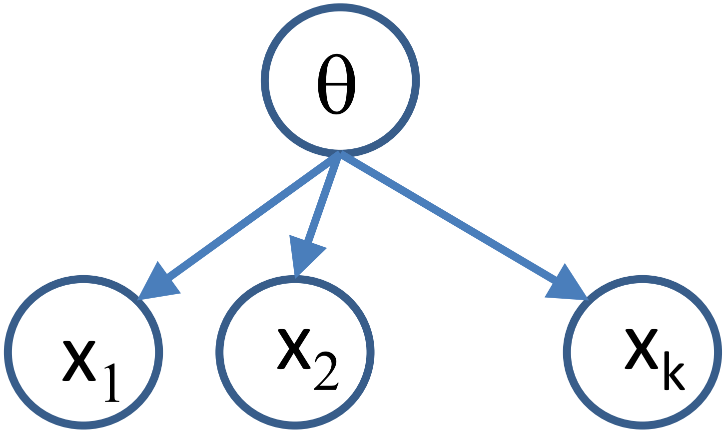
    for(i in 1:K){ X[i] ~ dbin(theta , N[i]) }
  - Every definition within ”for(i in 1:K){ }” should have index i on the left-hand-side of ”~” or ”←”.
  - Can make several nested loops, for ”x[i,j]” etc.
  - Can use nested indexing, for ”x[y[i]]”.
  - Can use arithmetics in indexing, for ”x[i+10]”

BUGS modeling

- Separate parameters example DAG:



- Common parameter example DAG:



BUGS language

- What distributions and logical functions are available?
  - Check the list from manual/menu.
  - Pay attention to parameterization!
  - A very useful function: step(). This can be used to create indicator variables, to compute probabilities by computing mean of the indicator.
  - What you define should be logically correct and computable in all situations. 1/X should never become 1/0 if X is stochastic.

BUGS language

- Data formatting (every data variable should appear in the model code)

```
list(x=4,
     y=c(3.5,7.2,9.1),
     z=structure(
          .Data=c(7,3,5,1,8,2),
          .Dim=c(2,3))     )
```

Alternative format

```
z[,1]   z[,2]   z[,3]
7    3    5
1    8    2
END
```

(Note: empty line after END)

BUGS language

- Irregular data can be coded using NA for ”missing”

```
list( z=structure(
        .Data=c(7,NA,NA,
                9,6,3,
                2,NA,5),
        .Dim=c(3,3)))
```

BUGS would generate predictions for NAs.

Alternatively, use auxiliary indexing:

```
list(z=c(7,9,6,3,2,5),person=c(1,2,2,2,3,3))
```

BUGS language

- Transformations of original data values can be declared within model code

    yy <- log(y)

    yy ~ dnorm(mu,tau)

    Here y would be given in data listing.

- Can check your data values from ’info’ → ’node info’ → ’values’ (if you doubt what values were loaded).

Tips

- Always think it as a DAG. → hierarchical model structures.
- Fixed data value has to be assigned to a stochastic ”~” node in the model code, not ”<-”. The latter would make ’multiple deterministic definitions’ error.
- Ddistr(? , ?) ← Parameters, not expressions. Check parameterization !
- Test first with a small number of iterations to see how slowly it runs.
- Give constants in data, not within code.
- Separate clearly what’s data, what’s model.
- Use comments # there are never too many!

Tips

- Collect definitions logically into groups (priors, likelihoods, predictions), easier to read.
- Transformations of data can be defined within code.
- Use indexing, and nested indexing.
- Avoid multiple definitions (e.g. within for-loops!) they are syntax errors.
- Break long expressions into short ones (avoid ’logical expression too complex’ error)
- Pay attention to naming of parameters, variables. They should be meaningful at first sight. (or write good explanations in comment lines)

Tips

- Constants cannot be monitored, but can check them from the node-info menu button.
- Sooner or later, it will be more convenient to run BUGS from R, try it later...
- For the inbuilt convergence diagnostics, you should pick overdispersed starting values for at least 3 chains.
- Think of identifiability: are there sufficient data? Is something hanging completely from prior? It is deceptively easy to build castles in the clouds….
- Make use of inprod to avoid writing long expressions a[1]*X[1]+a[2]*X[2]+… … . And make use of other useful functions available (see manual→ functions).

Tips

- **Finally:** don't get confused of what is (fixed) data, and what is unknown parameter. It is **always** about computing posterior distribution of the unknowns, conditionally on the known data:

P(unknowns | known data)

This posterior distribution is what you get from BUGS.