Parameter estimation in Physiological models using WinBUGS

Aris Dokoumetzidis
University of Athens

Structure

1. Introduction and the basics of WinBUGS

- Introduction to WinBUGS
- Familiarisation with the interface
- Winbugs language: models, priors, initial values
- Running Winbugs and the format of the results
- Chain convergence and diagnostics
- Running a simple BUGS mode

2. Fitting models to kinetic data

- Running a pharmacokinetic model
- Using scripts and interfacing WinBugs from other programmes
- Fitting ODE models to data

Structure

3. Hierarchical models

- Fitting population models to kinetic data
- Descriptive covariates
- Multi-level hierarchical models

4. Informative Priors

- Defining informative priors from previous WinBUGS runs
- Defining informative priors from literature

6. Model comparison

WinBUGS

BUGS: Bayesian inference **U**sing **G**ibbs **S**ampling
A computer programme to perform Bayesian inference of parametric models using Markov Chain Monte Carlo (MCMC) method.

Different versions: WinBUGS, OpenBUGS (same but open source),
Unix command line based versions.
Can be run on Linux and Mac through WINE

Also interfaces from Matlab, S-Plus, R and other

Winbugs developed on a platform called Blackbox (Omeron, Switzerland) using a language called Component Pascal which is a non-ANSI Pascal dialect.

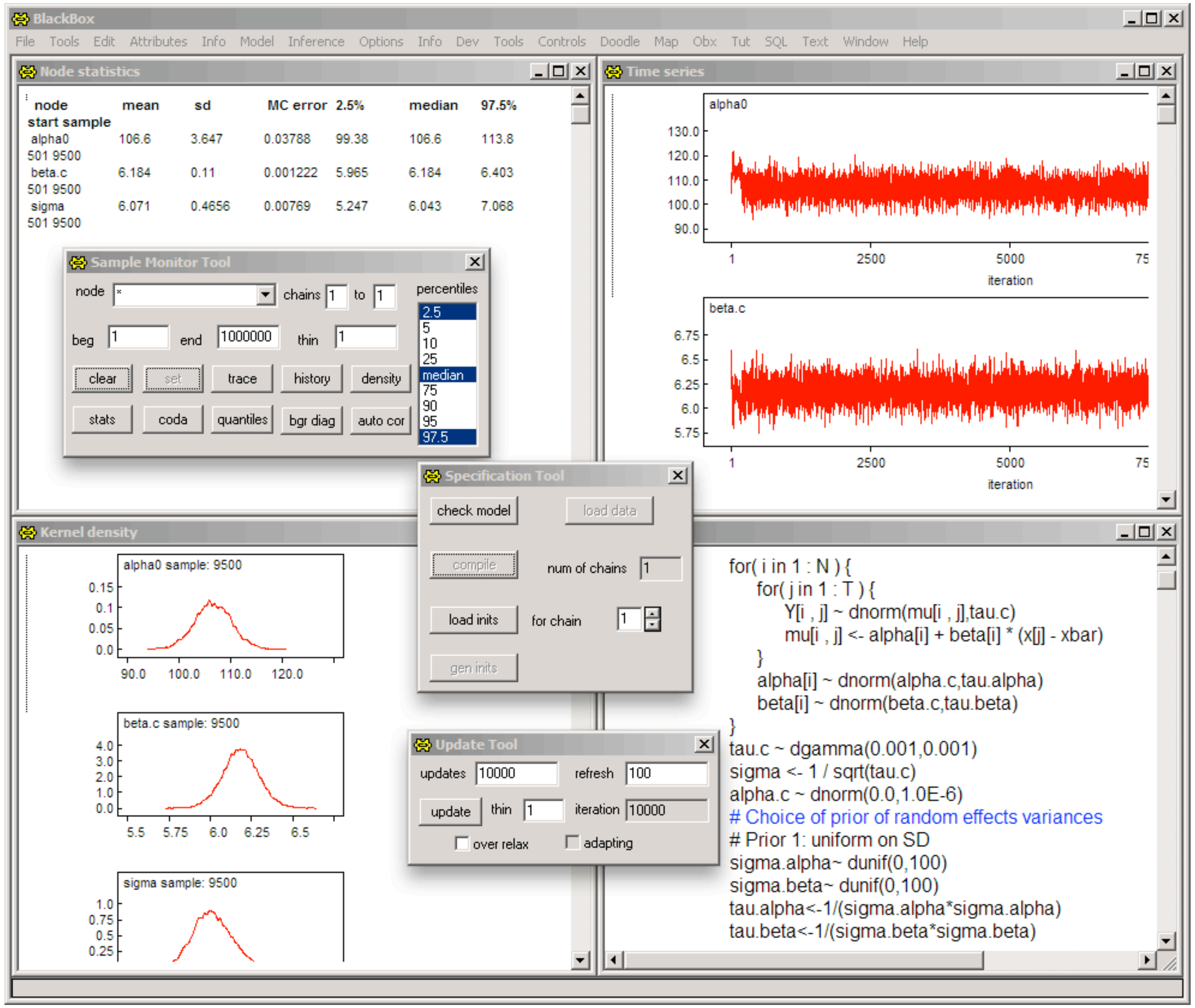
MCMC

Markov Chain Monte Carlo method creates random samples such that after enough iterations the chain converges to a stationary distribution which is the joint distribution of all stochastic parameters.
(Markov chain is a series of states with memory 1, i.e. each state depends only on the previous step)

Gibbs sampling: The chain is updated by sampling, for each variable, from its full conditional distribution where all other variables are considered known and are given the values of the previous state of the chain.

Metropolis – Hastings: Used when the distribution is not of known form like in nonlinear systems.
For each variable a new value is generated from a proposal distribution which is then compared with the old value. The new value is accepted with a probability so that the draws are actually simulating from the posterior distribution. If a value is rejected then the variable retains its old value.

WinBUGS interface



```
 node        mean    sd      MC error  2.5%    median  97.5%
start sample
 alpha0      106.6   3.647   0.03788   99.38   106.6   113.8
501 9500
 beta.c      6.184   0.11    0.001222  5.965   6.184   6.403
501 9500
 sigma       6.071   0.4656  0.00769   5.247   6.043   7.068
501 9500
```

Sample Monitor Tool: node * ; chains 1 to 1 ; beg 1 ; end 1000000 ; thin 1 ; percentiles 2.5, 5, 10, 25, median, 75, 90, 95, 97.5

Specification Tool: num of chains 1 ; for chain 1

Update Tool: updates 10000 ; refresh 100 ; thin 1 ; iteration 10000 ; over relax ; adapting

```
for( i in 1 : N ) {
    for( j in 1 : T ) {
        Y[i , j] ~ dnorm(mu[i , j],tau.c)
        mu[i , j] <- alpha[i] + beta[i] * (x[j] - xbar)
    }
    alpha[i] ~ dnorm(alpha.c,tau.alpha)
    beta[i] ~ dnorm(beta.c,tau.beta)
}
tau.c ~ dgamma(0.001,0.001)
sigma <- 1 / sqrt(tau.c)
alpha.c ~ dnorm(0.0,1.0E-6)
# Choice of prior of random effects variances
# Prior 1: uniform on SD
sigma.alpha~ dunif(0,100)
sigma.beta~ dunif(0,100)
tau.alpha<-1/(sigma.alpha*sigma.alpha)
tau.beta<-1/(sigma.beta*sigma.beta)
```

Running WinBUGS

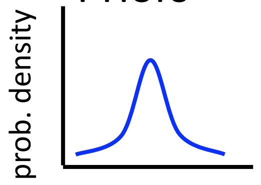
- Fit parametric models to data and obtain parameter estimates
- Parameter estimates typically are point estimates and uncertainty
- In WinBUGS the output is a chain of samples from a nonparametric distribution representing the posterior distribution
- With descriptive statistics on the chains, meaningful output is calculated
- Also being Bayesian one has to set priors in the form of parametric distributions

Running WinBUGS

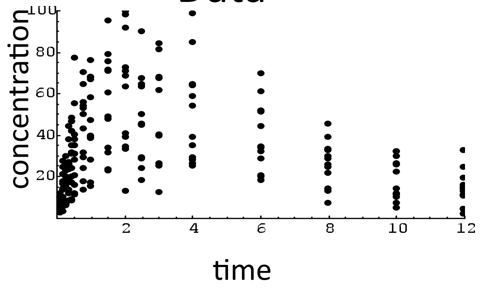
BUGS model

```
model {
   for (i in 1:n.ind) {
      for (j in ...
```

Priors



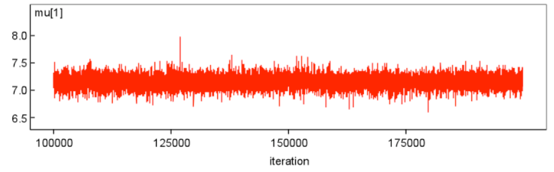
Data



Initial values

run model

posteriors



WinBUGS Language

Used to define the model and the priors.
Not really a full featured programming language

Can do:

- Assignments of variables. So called logical nodes: `a <- b`
- Definition of stochastic variables (nodes): `a ~ dnorm(mu,tau)`
- For-loops: `for( i in 1:N ) {` *<commands>* `}`
- Evaluate algebraic formulas (but big ones are slow, these need to be hardcoded and be called as functions): `a <- b + (exp(d) – 1)`
- No “if” statement but a “step” function can potentially serve the “if“ functionality for certain cases. e.g. to define branched expressions
- Supports indices and tables: `x[i, j, k]` or `y[1:5, 1:5]` or
`z[]` means all values of z and `z[,3]` means the 3rd column
- A definition of the data as a stochastic node is always present

Data

Data in WinBUGS are inserted as multidimensional matrices or vectors in 2 formats

Rectangular array

```
a[] b[]
1   1.2
2   1.6
3   0.12
END
```

S-Plus format

```
list(
a = c(1, 2, 3),
b = c(1.2, 1.6, 0.12),
m = structure(.Data=c(1,2,3,4,5,6,NA,8,9),.Dim=c(3, 3))
)
```

Priors

- Initial belief for the values of the parameters
- It’s in the form of distributions with expected values and dispersion accounting for uncertainty
- Always there, even when non-informative
- Defined in the model and are parametric distributions
- Usual such distribution are

```
a~dnorm(prior.mean, prior.prec)

a[1:N]~dmnorm(prior.mean[], prior.prec[,])

b~dgamma(alpha, beta)

b[1:N, 1:N]~dwish(sigma[,], deg]

f=dunif(init,fin)
```

Non-informative priors

- When no prior information is available we need to use non-informative priors
- Theoretically the estimation problem then takes into account only the likelihood from the data and becomes equivalent to a frequentist approach
- Flat distributions or as flat as possible, with correct properties
- Examples

`a~dnorm(1, 0.000001)` small precision -> high variance

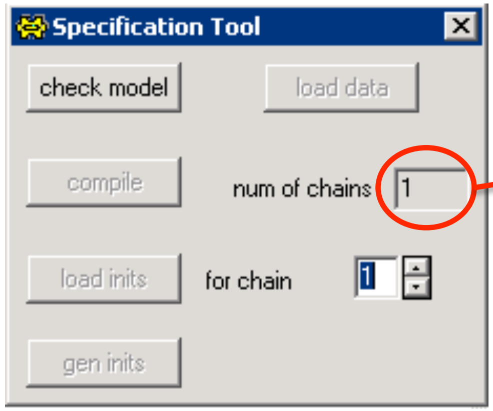
`b~dgamma(0.001, 0.001)` small equal values -> mean=a/b=1, variance=a/b^2 = 1000, can use uniform instead.

`b[1:N, 1:N]~dwish(sigma[,], N]` deg=size, least informative Wishart but not completely non-informative, we use it for variance parameters because it is positive definite.

`f=dunif(0,100)`

Initial values

- Initial values for all stochastic nodes are needed
- Random ones can be generated which is OK when informative priors exist
- With non-informative priors, reasonable initial values are necessary to initialise the chains
- They are defined as a list in S-Plus form, exactly as the data
- We can run more than one chains by having more sets of initial values



num of chains 1

Change this before pressing compile

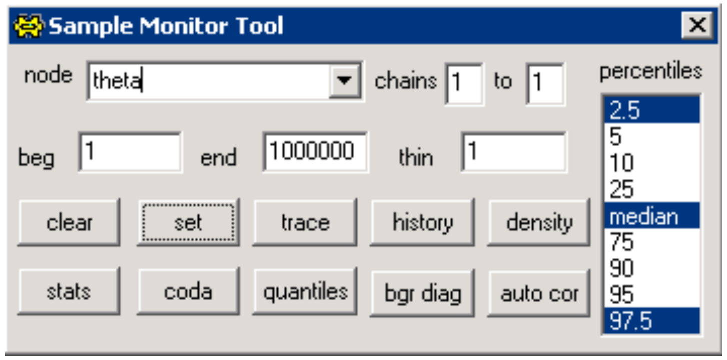
Running WinBUGS

Having defined the model, priors, data and inits, we define the parameters to be monitored.

Not all parameters need to be monitored, only the ones we are interested in

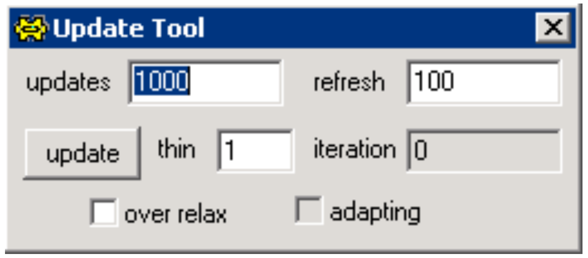
We start the chain (or chains) for a predefined number of iterations

After it's done we look at the results


Sample Monitor Tool

node theta chains 1 to 1 percentiles

beg 1 end 1000000 thin 1


Update Tool

updates 1000 refresh 100

thin 1 iteration 0

The results in WinBUGS

- Results come in the form of Monte Carlo samples which are hopefully drown from the correct posterior distribution.

- In principle we have to run the chains for long enough for this to happen

- There is no automatic way for termination of WinBUGS updating

- Chains must:

Converge after some initial burn-in iterations
Mix enough and sample representatively from the posterior distribution

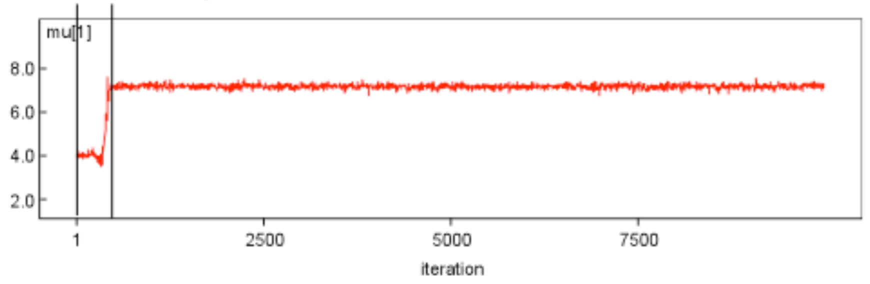
- Problem is that often chains mix slowly and have to be run for longer

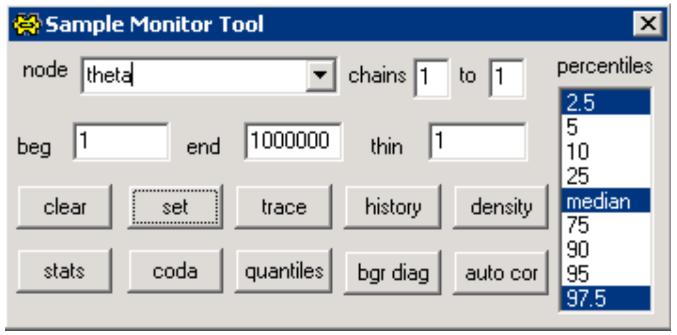
- Best way to tell is to look at the chains

- But formal methods exist to check

The results in WinBUGS

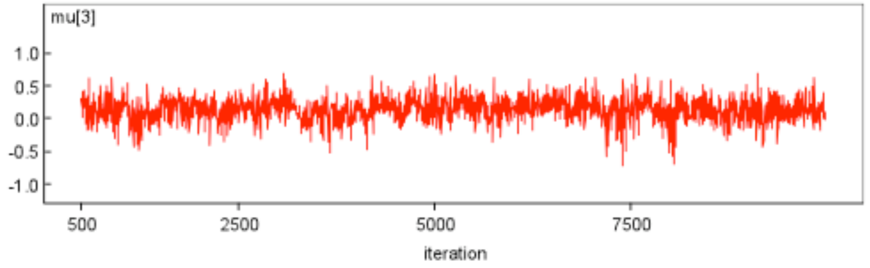
Initial burn in part has to be discarded



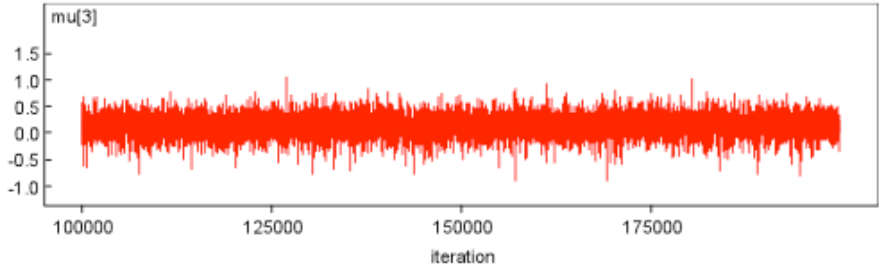


node theta
chains 1 to 1
beg 1 end 1000000 thin 1
percentiles
2.5
5
10
25
median
75
90
95
97.5

Chain with unwanted patterns, have to run longer



The same chain run for 10 times longer

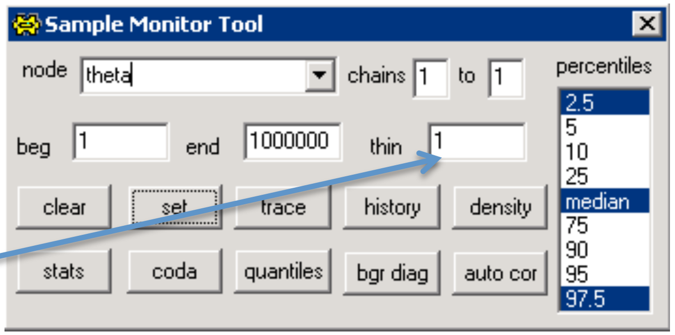


The results in WinBUGS

MCMC chains suffer from memory effects which are expressed by the autocorrelation plot and account for the patterns.

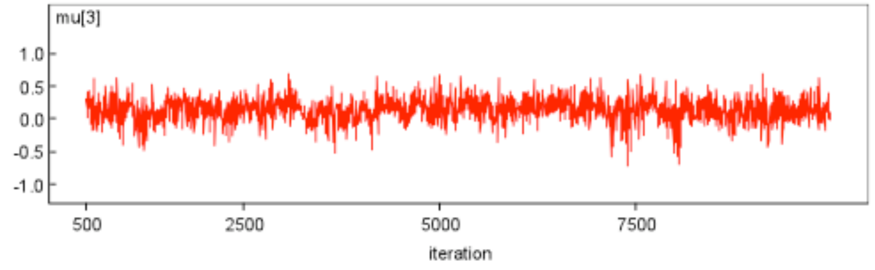
To address that you have to run for longer

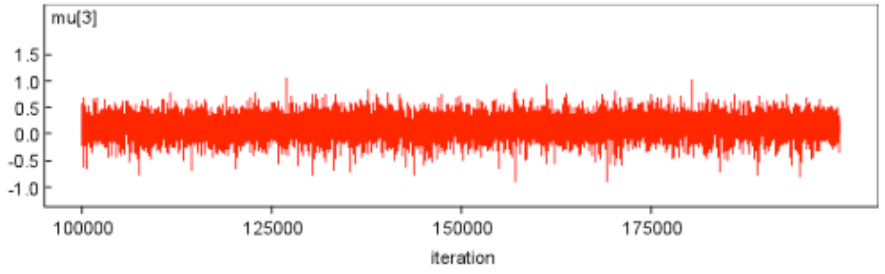
Also thin the chains


Sample Monitor Tool

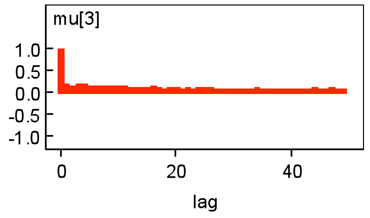
node theta | chains 1 to 1 | percentiles

beg 1 | end 1000000 | thin 1

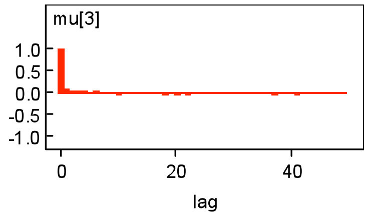


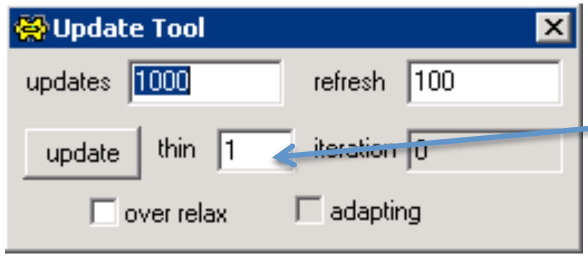


Autocorrelation plot



No autocorrelation




Update Tool

updates 1000 | refresh 100

thin 1 | iteration 0

Thin in update tool to save memory

Chain convergence

- Convergence to a marginal distribution
- Important to converge and also to sample representatively from posterior
- Most usual way to check is the visual inspection. Look for “fat hairy caterpillars” in the history plot
- Formal ways exist. The software CODA (for s-plus and R) implements several criteria
- Brooks-Gelman-Rubin (BGR) is available from within WinBUGS and is based on comparing at least 2 chains

BGR diagnostic

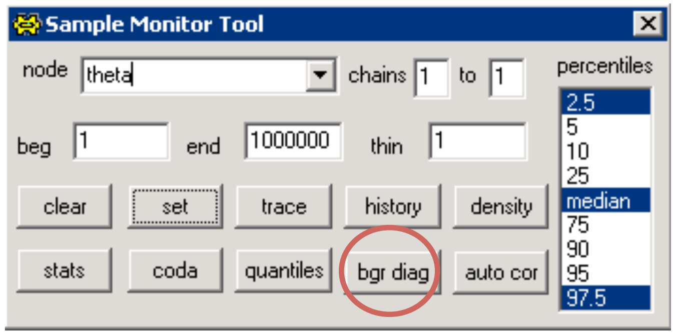
•At least 2 chains with differing starting values

•**Green**: width of 80% intervals of pooled chains. *Should be stable*

•**Blue**: Average width of 80% intervals for chains *Should be stable*

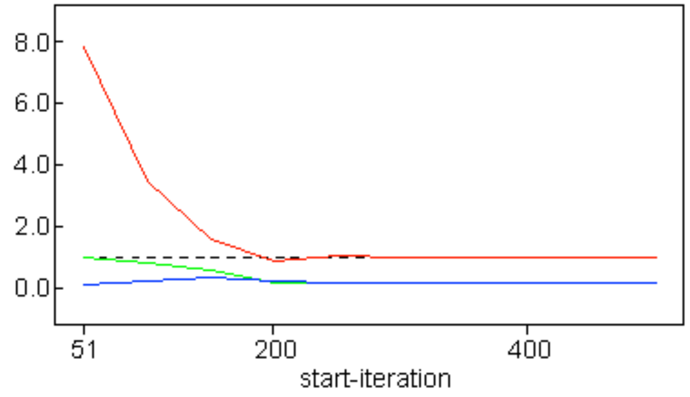
•**Red** (bgr): ratio of pooled/within. *Should be 1*

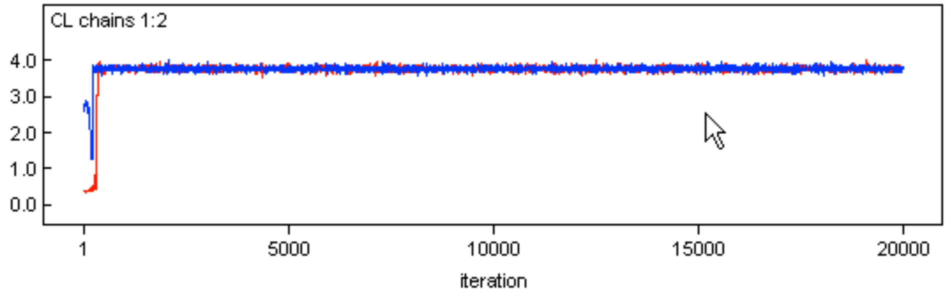
•Double click on plot and CTRL+right click gives statistics

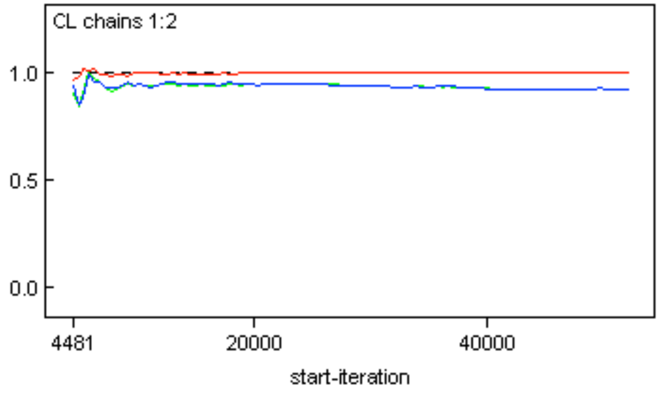

Sample Monitor Tool

node: theta | chains: 1 to 1

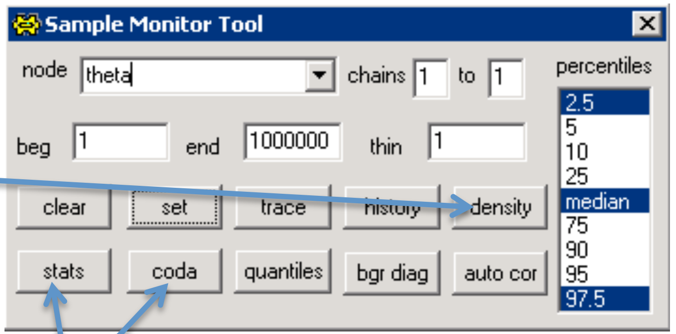
beg: 1 | end: 1000000 | thin: 1







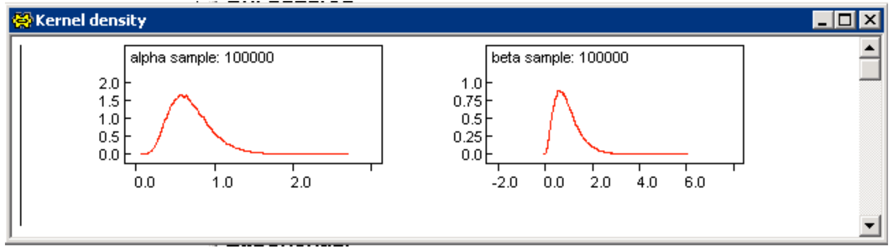
Having achieved decent chains one can:


Sample Monitor Tool

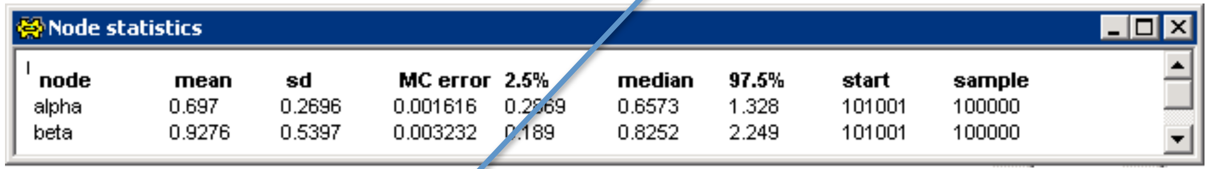
node theta chains 1 to 1

beg 1 end 1000000 thin 1

- Plot the density kernel plots or histograms



- Look at descriptive statistics of the chains that summarise the chains in meaningful means, SD, percentiles

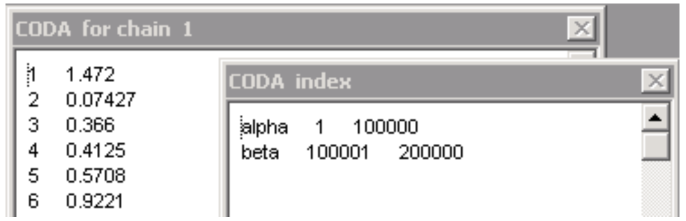

Node statistics

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
alpha	0.697	0.2696	0.001616	0.2969	0.6573	1.328	101001	100000
beta	0.9276	0.5397	0.003232	0.189	0.8252	2.249	101001	100000

- Export the chains using CODA (Coda is an S-plus and R programme for Winbugs):

A window including the chains of all the parameters appended

An index window pointing at the start and end lines for each parameter


CODA for chain 1

```
1    1.472
2    0.07427
3    0.366
4    0.4125
5    0.5708
6    0.9221
```

CODA index

```
alpha    1        100000
beta     100001   200000
```

Example: Fitting a normal distribution to data (Switch to software)

Model

```
model{
for(i in 1:20){
        data[i]~dnorm(mu,tau)
        }
mu~dnorm(0,0.0000001)
tau~dgamma(0.001,0.001)

sigma <- 1/sqrt(tau)
}
```

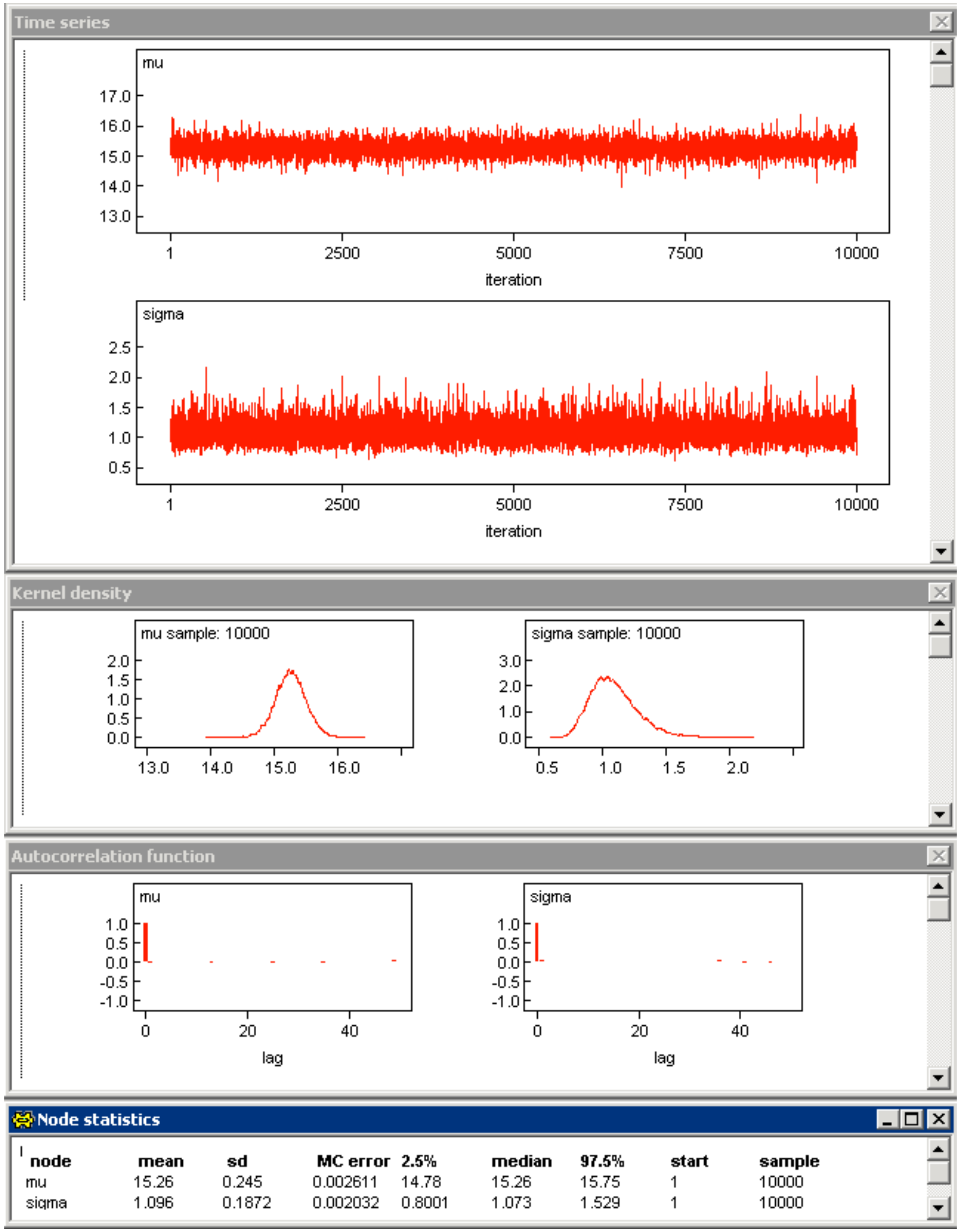
Non-informative priors

Data

```
list(
data = c(
14.29089337035724,16.83837707588466,15.19079787199498,13.82169101112563,15.23774011656273,
15.71770580130648,14.25348152327621,15.70644333836343,15.18221445171283,16.25817930399878,
14.07391242014187,15.09557349555977,15.80198510262376,14.53453436023399,15.88492175016683,
14.93469457573189,18.30022332988798,14.37724022490213,14.86854407159346,14.79744370388398)
)
```

Initial values

```
list(
mu=0,
tau=1
)
```



```
node      mean    sd      MC error  2.5%    median  97.5%   start   sample
mu        15.26   0.245   0.002611  14.78   15.26   15.75   1       10000
sigma     1.096   0.1872  0.002032  0.8001  1.073   1.529   1       10000
```

Simulate the data

Model

```
model{
for(i in 1:20){
		data[i]~dnorm(mu,tau)
		}
mu~dnorm(15,100000000)
tau~dgamma(100000,100000)

}
```

Extremely informative priors

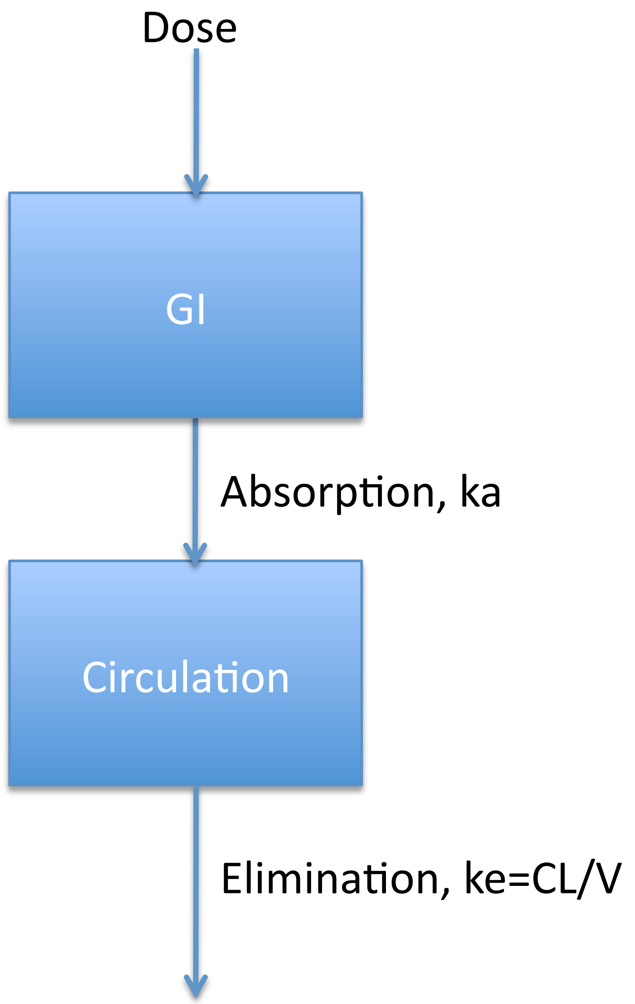
Data (empty)

```
list(
data=c(NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,
NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA,NA)
)
```

Summary -intro

- WinBUGS language to define the model
- Priors and initial values for the chains also needed
- Non-informative priors when no information is available give results close to a frequentist approach
- Tricky bit is how tell you 've run the chains for long enough. No formal termination criterion
- Chains must converge, and after converging, mix enough
- Some criteria for that exist (CODA) but safer is to rely on experience and visual inspection of chains.
- No Auto-Correlation, and “Hairy fat caterpillars” is what you want

Example: Fitting a simple pharmacokinetic model to data



Parameters

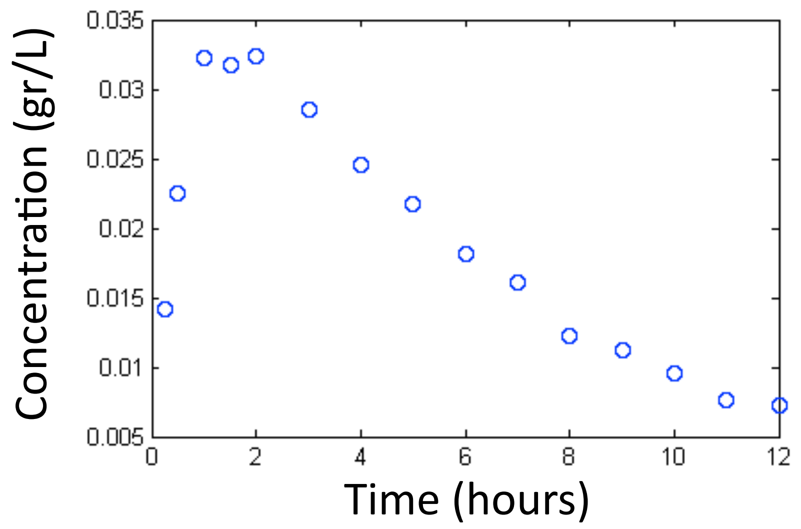
CL: clearance (3.75 L/h)

V: volume of distribution (25 L)

Ka: absorption first order rate constant (2 h^{-1})

$$f = \frac{Dose \cdot k_a}{V \cdot k_a - CL}\left(\exp\left(-\frac{CL}{V}t\right) - \exp(-k_a t)\right)$$

$\text{conc}_i = f_i * \exp(\varepsilon_i)$ $\qquad \varepsilon_i \sim N(0,\sigma)$



```
model{
for(i in 1:n){
        conc[i]~dnorm(logmodel[i],tau)
        logmodel[i]<-log(dose*ka/(V*ka-CL)*(exp(-CL/V*times[i])-exp(-ka*times[i])))
        }
CL<-exp(mu[1])
V<-exp(mu[2])
ka<-exp(mu[3])+CL/V

mu[1:3] ~dmnorm(prior.mean[],prior.prec[,])

tau~dgamma(0.001,0.001)
sigma<-1/sqrt(tau)
}

list(
n=15,
dose=1,
times=c(0.25,0.5,1,1.5,2,3,4,5,6,7,8,9,10,11,12),
conc=c(-4.25492,-3.79575,-3.43383,-3.45049,-3.43203,-3.55484,-3.70655,
-3.82849,-4.00805,-4.13308,-4.40292,-4.4919,-4.64878,-4.87452,-4.92813),

prior.mean=c(0,0,0),
prior.prec=structure(.Data=c(1.0E-6,0,0,0,1.0E-6,0,0,0,1.0E-6),.Dim=c(3,3)))
```

Logged model

Slow, should hardwire:
4 sec < 23 sec
for 100k iterations

Parameters also logged

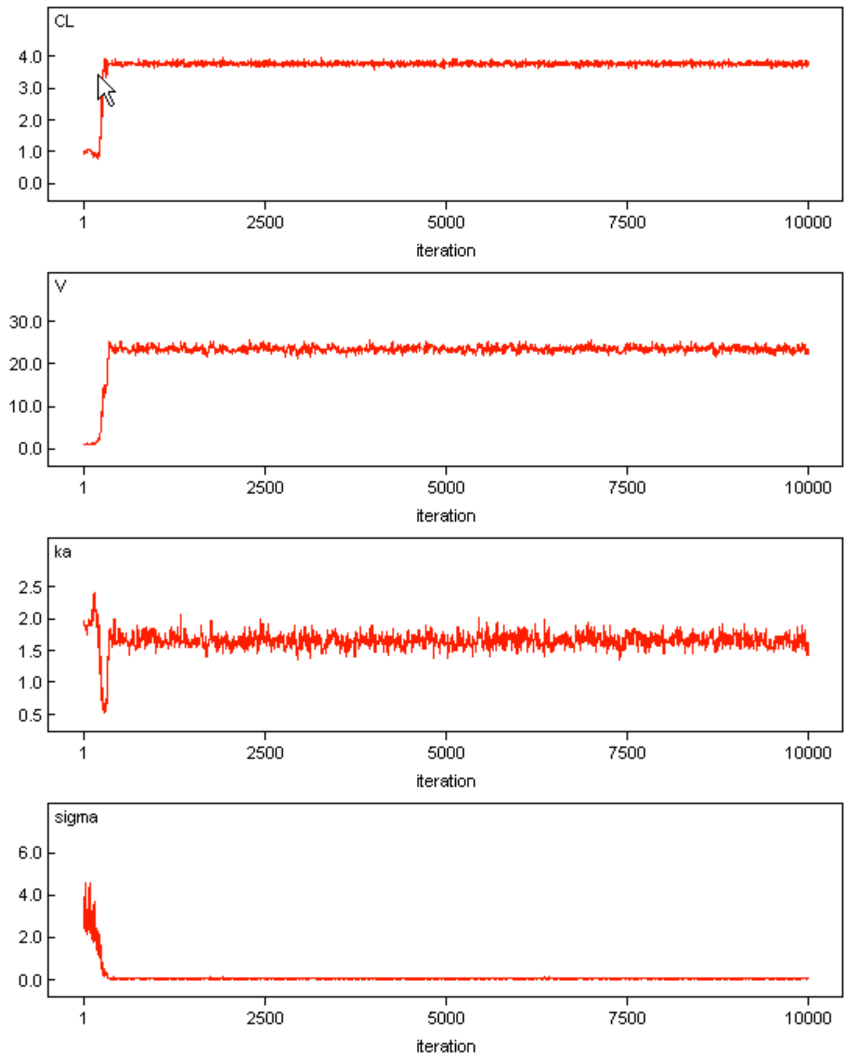
Local identifiability: parameterise as mu[3]=ka-ke, i.e. ka > ke

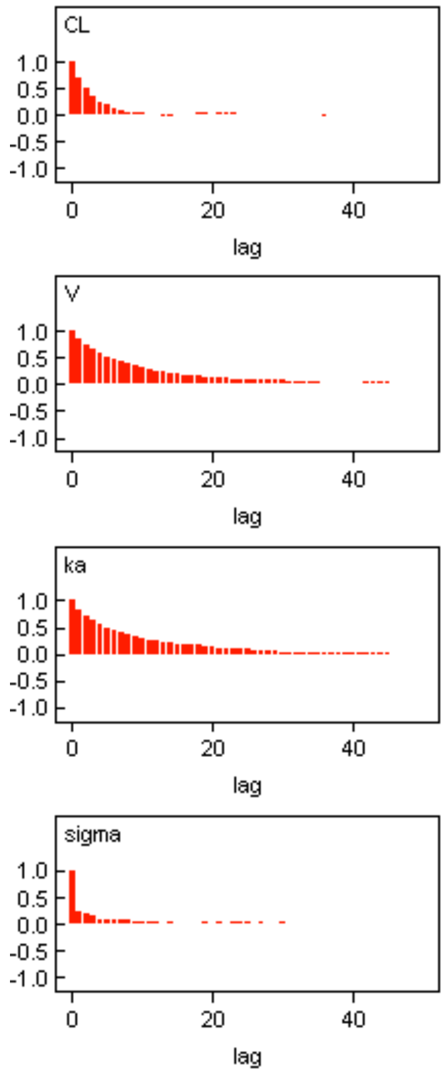
Correlations between parameters

Logged data

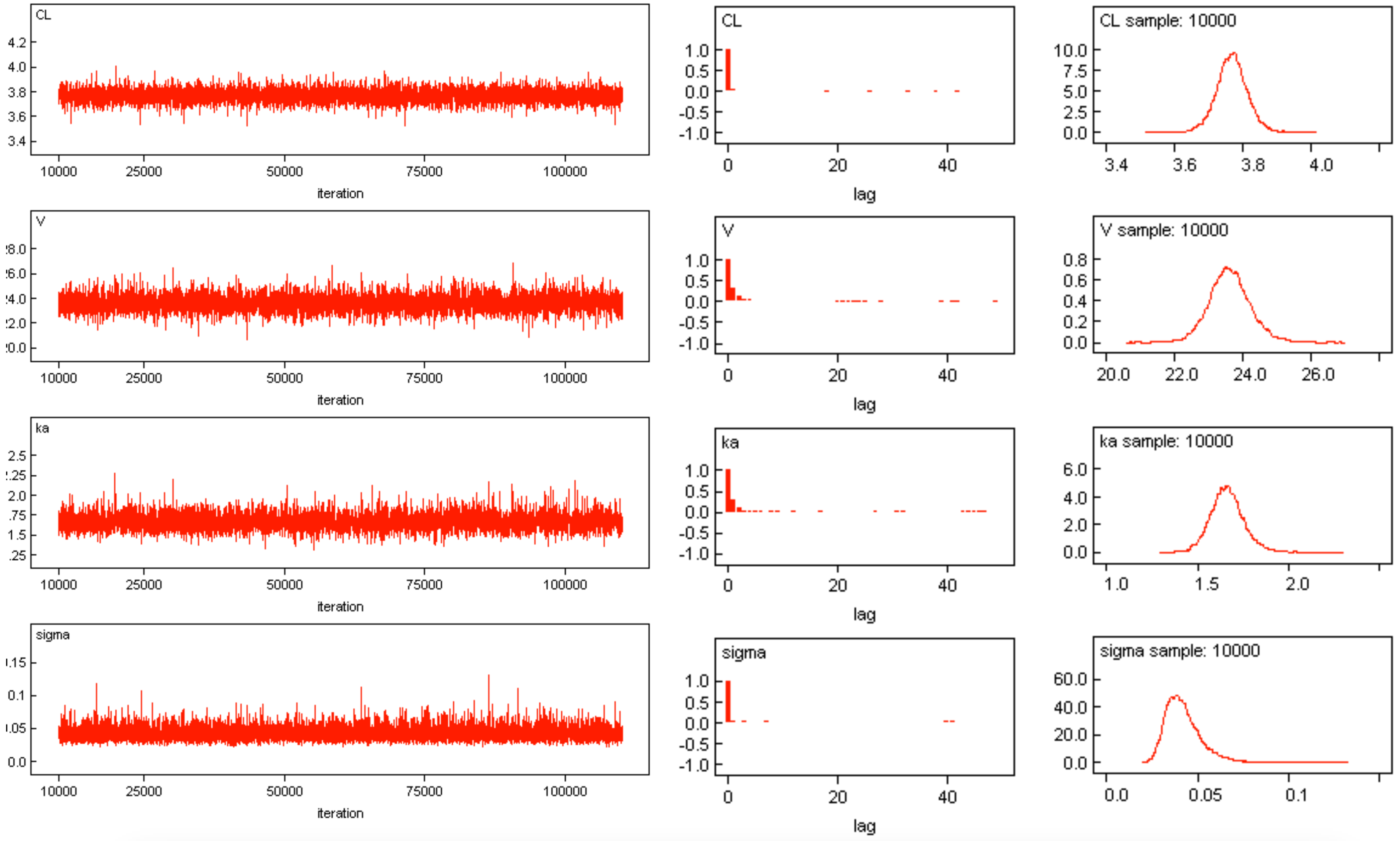
Non-informative priors

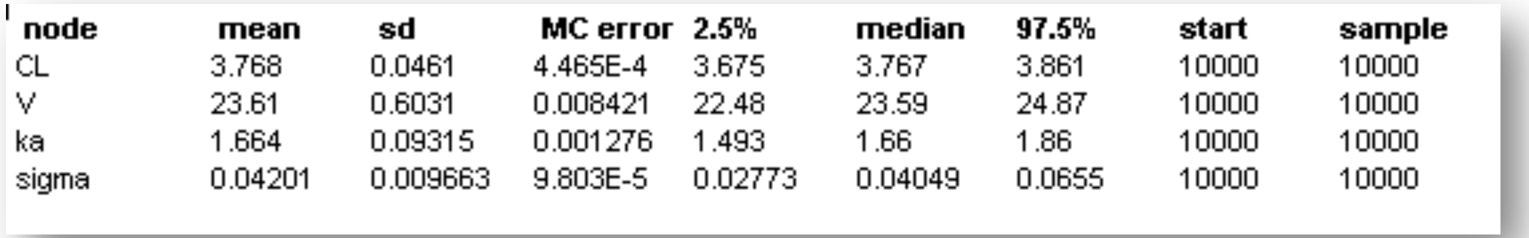
Results: run for 10k iterations





Results: run for another 100k iterations and keep 1:10, (took 23 secs to run)

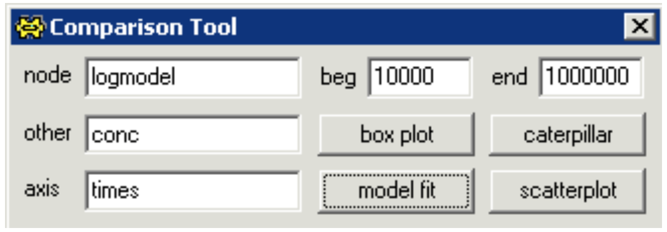




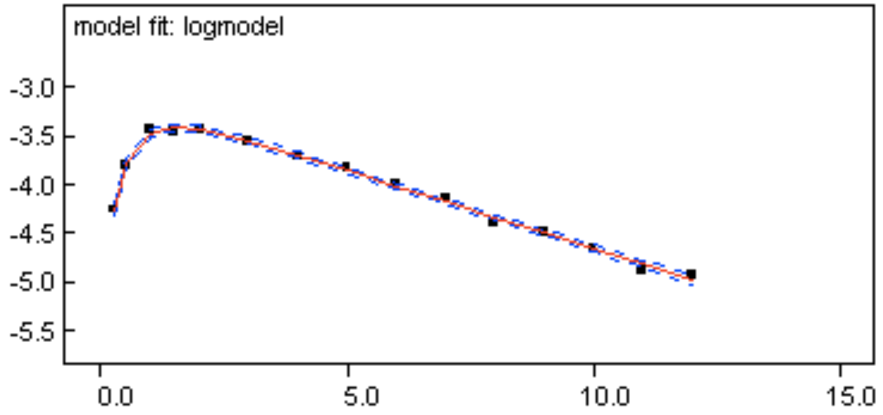
node	mean	sd	MC error	2.5%	median	97.5%	start	sample
CL	3.768	0.0461	4.465E-4	3.675	3.767	3.861	10000	10000
V	23.61	0.6031	0.008421	22.48	23.59	24.87	10000	10000
ka	1.664	0.09315	0.001276	1.493	1.66	1.86	10000	10000
sigma	0.04201	0.009663	9.803E-5	0.02773	0.04049	0.0655	10000	10000

Results

A plot of the model with the data can be obtained from the ***Comparison Tool*** found in the ***Inference*** menu



node: logmodel beg: 10000 end: 1000000
other: conc
axis: times



Median (-), 2.5% and 97.5% percentiles (--) are plotted

Have to make sure “logmodel” is monitored

Parameter space

- Physiological parameters usually are positive

- A reasonable assumption is that they follow lognormal distributions

- To implement that we work on the log scale for the parameters

log-parameter ~ N(mean, var)
and exponentiate **parameter <- exp(log-parameter)** to feed into the model

- Informative priors may be easily defined. Any non-infinite (effectively) variance means informative prior.

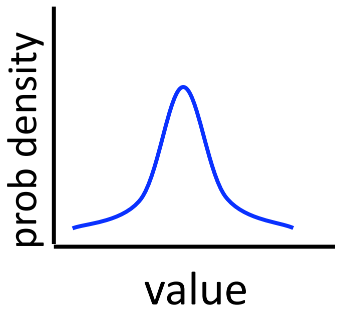
- Non-informative prior: infinite variance i.e. zero precision

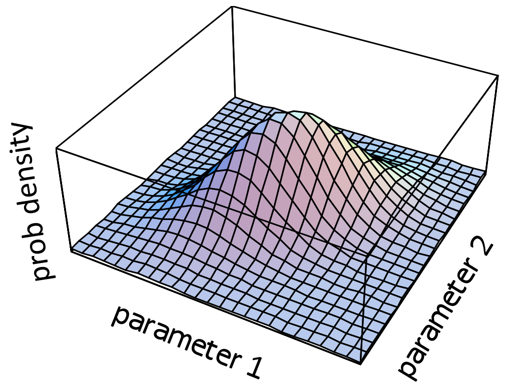
- Informative priors directly influence the result according to Bayes' theorem

i.e. **posterior = prior * likelihood**, (normalised to integrate to 1)

Parameter space

- Parameters are considered to follow multivariate distributions. i.e. are correlated.
- Basically means that the chains should be treated in pairs, triplets and so on





- The stats option in WinBUGS does not report covariance terms but they exist.
- To calculate them, should export the chains and perform multivariate descriptive statistics externally

Residual error: model

Least Squares: weights: 1 or $1/y^2$

Likelihood methods: a random effect is considered for error with specific distribution

Usual types of residual error:

Error type	Error model	distrib	WinBUGS
Additive error	$y_i = f_i + \varepsilon_i$	$\varepsilon_i \sim N(0,\sigma)$	straightforward
Proportional error	$y_i = f_i\,(1+\varepsilon_i)$	$\varepsilon_i \sim N(0,\sigma)$	NA
Exponential error (similar to proportional)	$y_i = f_i \, {*}\exp(\varepsilon_i)$	$\varepsilon_i \sim N(0,\sigma)$	Log model and data. Makes it additive on the log scale
Combined error:	$y_i = f_i\,(1+\varepsilon_{1,i}) + \varepsilon_{2,i}$	$\varepsilon_i \sim N(0,\Sigma)$	feasible

Also other types…

Residual error: prior

As in all variance terms of normally distributed variables residual error can be considered to be inverse-Gamma distributed

Precision (tau=1/var=1/SD^2) is Gamma distributed

Gamma distribution Gamma(a,b) has **mean=a/b** and **var=a/b^2**

Appropriate informative priors may be defined based on that

Non-informative priors:

```
tau ~ dgamma(0.001, 0.001)
sigma <- 1/sqrt(tau)
```

Concerns have been raise that this may not be entirely non-informative

Alternatively uniform on sigma (recommended by WinBUGS authors):

```
sigma ~ dunif(0, 100)
tau <- 1/(sigma*sigma)
```

But also possible to use uniform on variance or tau

Structural identifiability

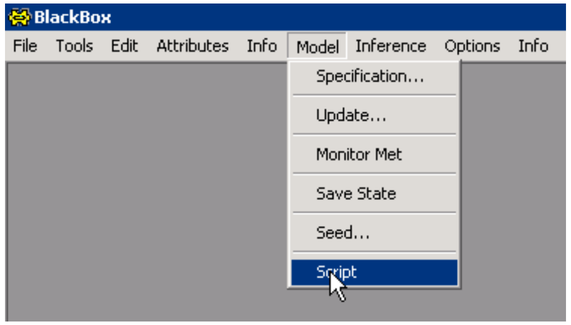
- A parameter estimation problem should ideally be globally identifiable
- Each profile is produced by a unique set of parameter values
- Some systems are only locally identifiable due to structural symmetries meaning that each time a profile is produced by 2 or more different sets of parameter values
- In a Levenberg – Marquardt algorithm (local optimiser) different solutions are obtained for different initial values
- But in MCMC the entire parameter space within the prior is explored, therefore the system should be made globally identifiable by setting constraints or appropriate parameterisation
- In our example parameterising as **mu(3)=ka-ke > 0** (log-normal) makes the problem globally identifiable. But care must be taken as **ka<ke** is also physically plausible

Using scripts

Clicking buttons is not everybody’s favorite way of using software and is not practical when automation is needed.

WinBUGS supports simple scripts which can execute the needed tasks as a sequence of text commands

```
display('log')
check('lipari/ex2model.txt')
data('lipari/ex2data.txt')
compile(1)
inits(1, 'lipari/ex2inits.txt')
gen.inits()
set(CL)
set(V)
set(ka)
set(sigma)
update(100000)
stats(*)
history(*)
density(*)
autoC(*)
save('lipari/ex2out')
CODA(*, codaout)
```



Interfacing with other software

Several packages have been developed making possible to invoke a WinBUGS run and retrieve the results through other softwares

Such as:

Package	Platform	Features
R2Winbugs	R	
MATBUGS	MATLAB	
BugsXLA	Excel	No WinBUGS knowledge required
PyBUGS	Python	Computer cluster
bugsParallel	R	Computer cluster

And others …

They all work by building a script file, running WinBUGS as a shell command and then importing a CODA file with the results

Hardwiring functions in WinBUGS

- Winbugs handles complex algebraic expressions.
- However anything but the very basic of models, is very slow
- It is better to hardwire and pre-compile even relatively simple models
- Example of PK Winbugs model

```
logmodel[i]<-log(dose*ka/(V*ka-CL)*(exp(-CL/V*times[i])-exp(-ka*times[i])))
```

Can be written as

```
logmodel[i]<-PK1abs1m(mu[],dose,times[i])
```

Where `PK1abs1m()` is a compiled function used much like `log()` or `exp()`

- Improvement of nearly 6 times in this example

(4 secs vs 23 secs for 100k interations)

- This is done in ***Blackbox*** software which is the platform in which WinBUGS itself is developed and written in ***Component Pascal***

Hardwiring functions in WinBUGS

- Install Blackbox, download it first from http://www.oberon.ch/blackbox.html
- Copy WinBUGS in the Blackbox directory and run Blackbox
- This runs WinBUGS from within Blackbox
- Download and install WBDev package from WinBUGS development site
- Use templates and instructions provided to write you own functions in Component Pascal. You don't really need to have Pascal knowledge
- Compile function by pressing ^K
- Add a line for the definition of the function in the ***grammar*** file ([root]\WBDev\Rsrc\Grammar.odc):

```
s <- "PK1abs1m"(v, s, s)  "WBDevPK1abs1m.Install”
```

- Function ready to use in WinBUGS

Component Pascal example

```
(*1*)   MODULE WBDevPK1abs1m;
                IMPORT
                        WBDevScalar,
(*2*)                   Math;
                TYPE
                        Function = POINTER TO RECORD (WBDevScalar.Node) END;
                        Factory = POINTER TO RECORD (WBDevScalar.Factory) END;
                VAR
                        fact-: WBDevScalar.Factory;
(*3*)           PROCEDURE (func: Function) DeclareArgTypes (OUT args: ARRAY OF CHAR);
(*4*)           BEGIN
(*5*)                   args := "vss";
(*6*)           END DeclareArgTypes;

(*7*)           PROCEDURE (func: Function) Evaluate (OUT value: REAL);
(*8*)           CONST
(*9*)                   parameters = 0; dose = 1; time = 2;
(*10*)          VAR
(*11*)                  F,ke,cl,V,ka,D,t,c: REAL;
       )        BEGIN
                        cl:=Math.Exp(func.arguments[parameters][0].Value());
                        V:=Math.Exp(func.arguments[parameters][1].Value());
                        ke  := cl/V;
                        ka:=ke+Math.Exp(func.arguments[parameters][2].Valu
                        F:=1;
                        D:=func.arguments[dose][0].Value();
                        t:= func.arguments[time][0].Value();
                        c:= F*D*ka/((ka-ke)*V)*(Math.Exp(-ke*t)-Math.Exp(-ka
                        IF c > 0 THEN;
(*24*)                          value  := Math.L
                        ELSE
                                value:=-1000000;
(*25*)              END;
(*26*)          END Evaluate;
                PROCEDURE (f: Factory) New (option: INTEGER): Function;
                VAR
                        func: Function;
                BEGIN
                        NEW(func); func.Initialize; RETURN func;
                END New;
                PROCEDURE Install*;
                BEGIN
                        WBDevScalar.Install(fact);
                END Install;
                PROCEDURE Init;
                VAR
                        f: Factory;
                BEGIN
                        NEW(f); fact := f;
                END Init;
        BEGIN
                Init;
(*1*)   END WBDevPK1abs1m.
```

Parameters logged

Local identifiability: parameterise as mu[3]=ka-ke, i.e. ka > ke

Logged model

Press ^K to compile

Hardwiring functions in WinBUGS

- More than 1 templates exist to cover cases where more substantial modifications are needed, such as when the output is a vector

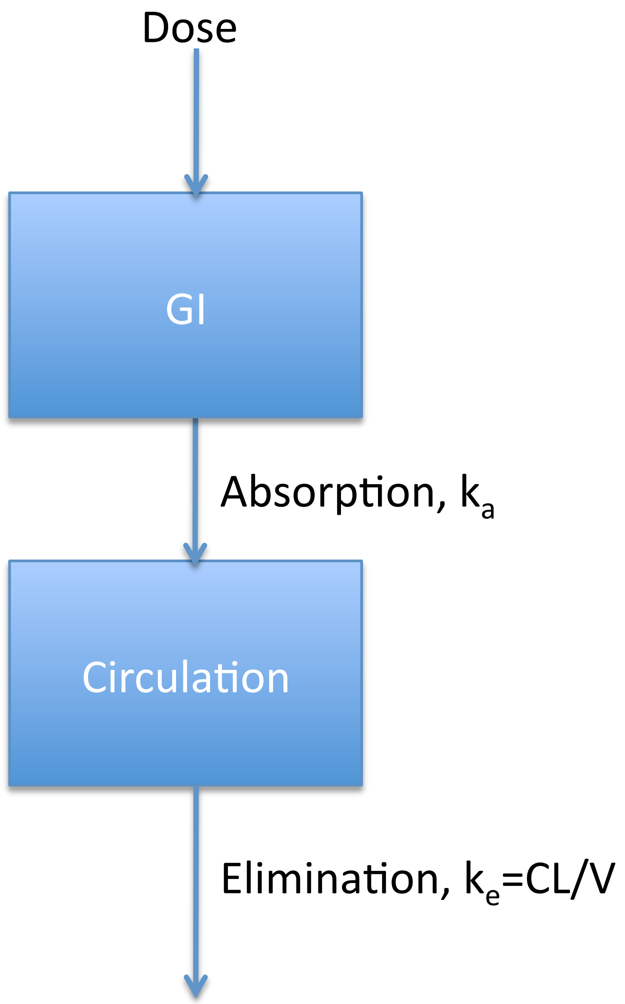
- Also different templates are provided for user-defined statistical distributions

- A completely separate package called ***WBDiff*** works in the same rationale and provides implementation of Runge – Kutta ODE solver

Model defined by ODEs in WinBUGS

- With the WBDev any arbitrary function can be calculated including one whose output is generated by internal iterations inside the Pascal routine.
- In this way an ODE or PDE may be solved and its output returned to Winbugs, buy implementing and ODE solver in Pascal.
- WBDiff is a package that implements Runge-Kutta 4th and 5th order which is a variable step non-stiff ODE solver
- Needs to be installed like the WBDiff but it is independent from it
- Allows specification of systems of ODEs within WinBUGS language for convenience (slow option)
- But also provides templates in Pascal for fully precompiled functions like in WBDev (faster option)

Example: Fitting a simple ODE pharmacokinetic model to data



Parameters
CL: clearance (3.75 L/h)
V: volume of distribution (25 L)
Ka: absorption first order rate constant (2 h^{-1})

$$\frac{dA_1}{dt} = -k_a A_1$$

$$\frac{dA_2}{dt} = k_a A_1 - k_e A_2$$

Initial conditions

$A_1(0)$=dose, $A_2(0)=0$

Observable is concentration in circulation

$C_b = A_2/V$

Include ODEs in coded in WinBUGS language

```
sol[1:n.grid,1:dim]<-ode(init[1:dim],grid[1:n.grid],D(C[1:dim],t),origin,tol)
```

Ode function
Initial conditions
Time points
Gradients
Initial time (0)
Tolerance (1E-3)

$$\frac{dA_2}{dt} = k_a A_1 - k_e A_2$$

```
D(A[2], t) <- ka * A[1] - ke * A[2]
```

WinBUGS code for ODE model

```
model{
for(i in 1:n){
     conc[i]~dnorm(logmodel[i],tau)
     logmodel[i]<-log(solution[i,2]/V)
}
solution[1:n, 1:2] <- ode(init[], times[], D(A[1:2], t), origin, tol)
D(A[1], t) <- -ka * A[1]
D(A[2], t) <- ka * A[1] - ke * A[2]

ke <- CL/V
CL<-exp(mu[1])
V<-exp(mu[2])
ka<-exp(mu[3])+CL/V

init[1] <- dose; init[2] <- 0

mu[1:3] ~dmnorm(prior.mean[],prior.prec[,])
tau~dgamma(0.001,0.001)

sigma<-1/sqrt(tau)
}
```

ODE hardwired

```
model{
for(i in 1:n){
     conc[i]~dnorm(logmodel[i],tau)
     logmodel[i]<-log(solution[i,2]/V)
}
solution[1:n, 1:2] <- one.comp.model(init[], times[], mu[], origin, tol)

ke <- CL/V
CL<-exp(mu[1])
V<-exp(mu[2])
ka<-exp(mu[3])+CL/V

init[1] <- dose; init[2] <- 0

mu[1:3] ~dmnorm(prior.mean[],prior.prec[,])
tau~dgamma(0.001,0.001)

sigma<-1/sqrt(tau)
}
```

Component Pascal code for ODE module

```
MODULE WBDiffOneCompModel;

       IMPORT
              WBDiffODEMath,
              Math;

       TYPE
              Equations = POINTER TO RECORD (WBDiffODEMath.Equations) END;
              Factory = POINTER TO RECORD (WBDiffODEMath.Factory) END;

       CONST
              nEq = 2;

       VAR
              fact-: WBDiffODEMath.Factory;

       PROCEDURE (e: Equations) Derivatives (IN theta, A: ARRAY OF REAL; n: INTEGER; t: REAL; OUT dAdt: ARRAY OF REAL);
       VAR
              i: INTEGER;
              ka, ke,cl, V: REAL;
       BEGIN
              cl:=Math.Exp(theta[0]);
              V:=Math.Exp(theta[1]);
              ke := cl/V;
              ka:=ke+Math.Exp(theta[2]);
              dAdt[0]:= -ka*A[0];
              dAdt[1]:= ka*A[0]-ke*A[1];
       END Derivatives;
```

● ● ●

Press ^K to compile

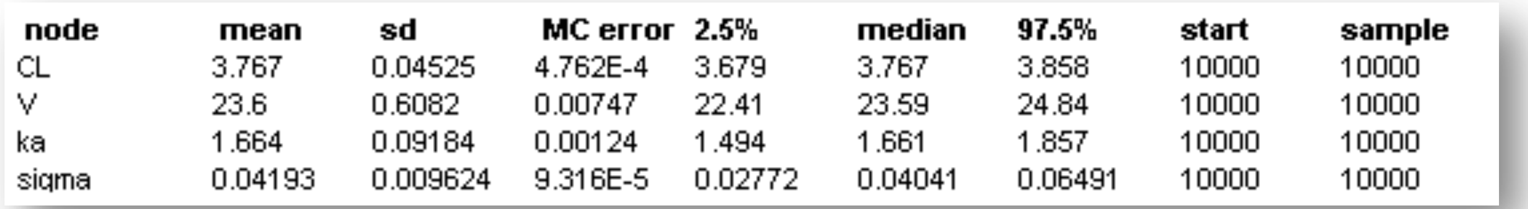
And include a line in the “Grammar.odc” file found in “[root]\WBDiff\Rsrc”

```
v <- "one.comp.model"(v, v, v, s, s)      "MathRungeKutta45.Install; WBDiffOneCompModel.Install"
```

ODE results

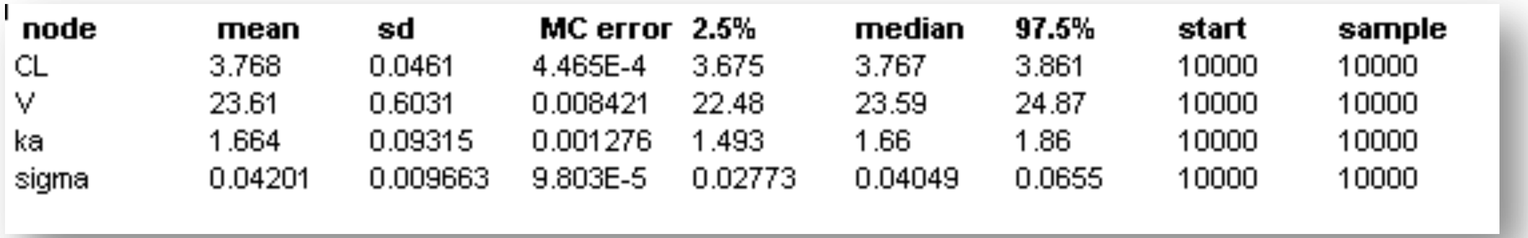
We obtain similar results but slower

ODE model in WinBUGS language 130 secs and hardwired 30 secs



node	mean	sd	MC error	2.5%	median	97.5%	start	sample
CL	3.767	0.04525	4.762E-4	3.679	3.767	3.858	10000	10000
V	23.6	0.6082	0.00747	22.41	23.59	24.84	10000	10000
ka	1.664	0.09184	0.00124	1.494	1.661	1.857	10000	10000
sigma	0.04193	0.009624	9.316E-5	0.02772	0.04041	0.06491	10000	10000

Analytical model in WinBUGS language 23 secs and hardwired 4 secs



node	mean	sd	MC error	2.5%	median	97.5%	start	sample
CL	3.768	0.0461	4.465E-4	3.675	3.767	3.861	10000	10000
V	23.61	0.6031	0.008421	22.48	23.59	24.87	10000	10000
ka	1.664	0.09315	0.001276	1.493	1.66	1.86	10000	10000
sigma	0.04201	0.009663	9.803E-5	0.02773	0.04049	0.0655	10000	10000

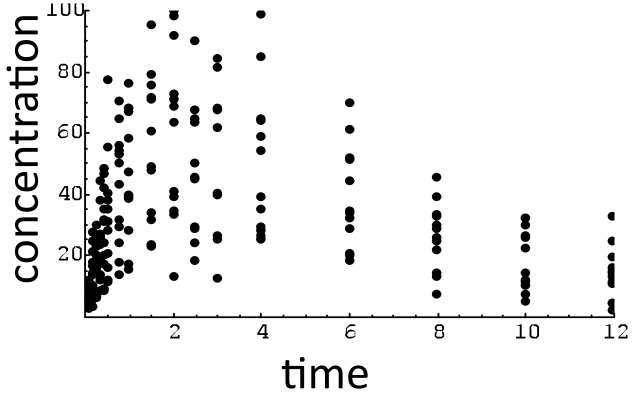
Issues with ODE solver

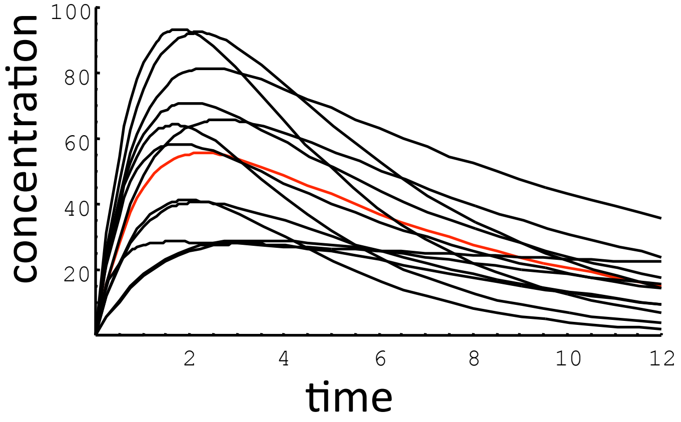
- Slow! Typically larger models are considered so it will be slower than the example shown
- More dimensions may need more iterations to mix properly, so even slower
- WBDiff ODE solver is for non-stiff problems
- Noninformative priors may cause problems as certain extreme combinations of parameters may produce numerical problems inthe ODE solver (Trap windows)
- Often informative priors are needed to avoid sampling problematic parameter values

Summary - kinetics

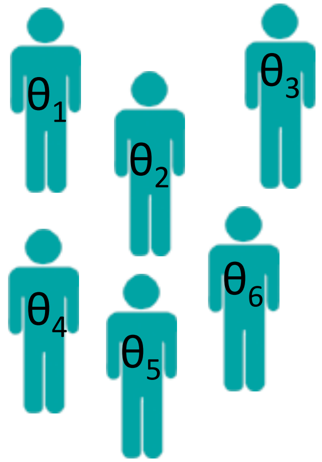
- Fitted a simple kinetic model to data
- Looked at chain convergence
- Parameter distribution is multivariate normal and often on the log scale
- Looked at different residual error models and prior for that
- Mentioned structural identifiability and why this is particularly important in MCMC
- Use scripts and call WinBugs from other programmes
- Hardwire computationally intensive expressions (nearly all but the simplest) to speed up calculations
- User ODE models, in WinBUGS language and hardwiring them
- Compared differences in speed for functions and ODEs in WinBUGS language and hardwired

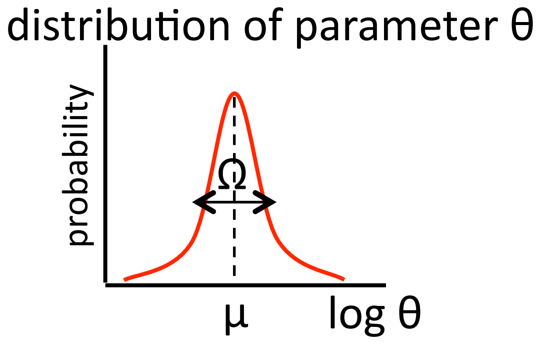
Hierarchical modes or population approach





inter- individual variability

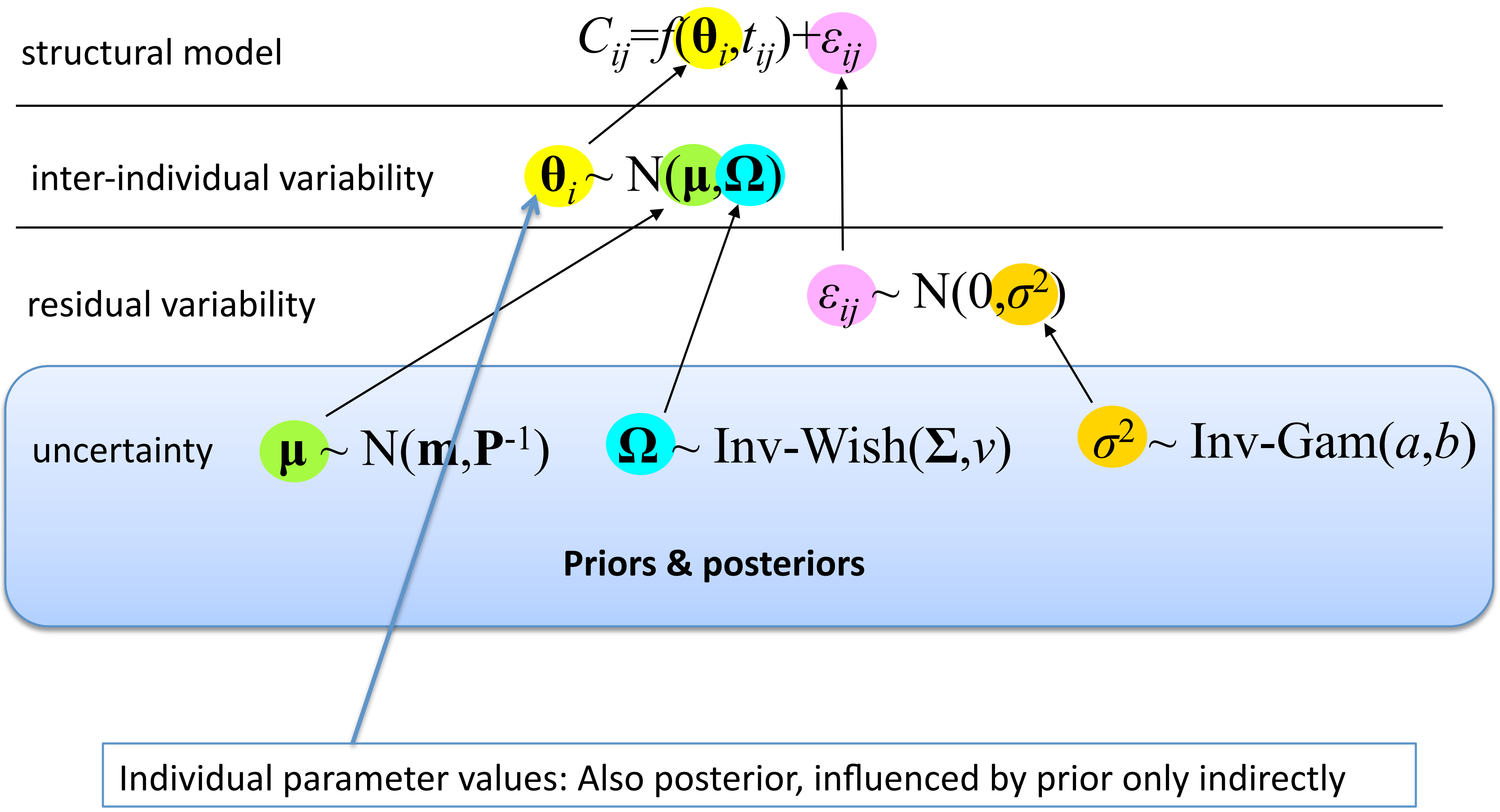




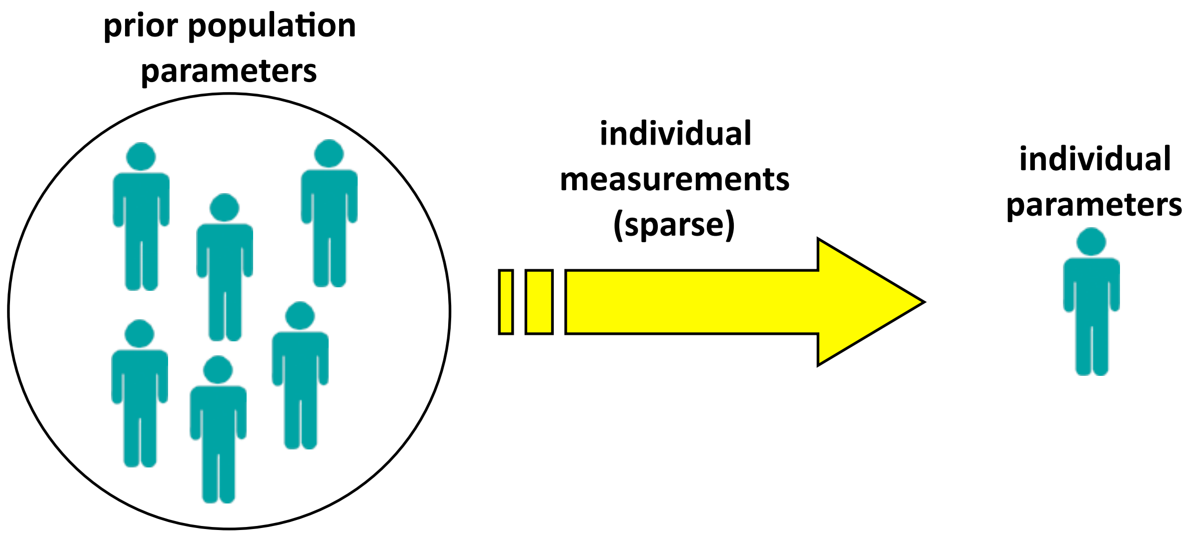
mean value: μ

variance: Ω

Hierarchical model (3 levels)


structural model
$C_{ij}=f(\boldsymbol{\theta}_i,t_{ij})+\varepsilon_{ij}$
inter-individual variability
$\boldsymbol{\theta}_i \sim \mathrm{N}(\boldsymbol{\mu},\boldsymbol{\Omega})$
residual variability
$\varepsilon_{ij} \sim \mathrm{N}(0,\sigma^2)$
uncertainty
$\boldsymbol{\mu} \sim \mathrm{N}(\mathbf{m},\mathbf{P}^{-1})$
$\boldsymbol{\Omega} \sim \text{Inv-Wish}(\boldsymbol{\Sigma},v)$
$\sigma^2 \sim \text{Inv-Gam}(a,b)$
Priors & posteriors
Individual parameter values: Also posterior, influenced by prior only indirectly

Bayesian individualization



Useful in estimating individual parameters with very little data

Applications: Therapeutic Drug Monitoring
Dose individualization

Fitting a PK model to population data

```
model {
        dose<-600000
        for (i in 1:n.ind) {
                for (j in off.data[i]:(off.data[i + 1] - 1)) {

                        data[j] ~ dnorm(model[j], tau)
                        model[j] <- PK1abs1m(theta[i,1:p],dose,time[j])
                }
                theta[i, 1:p] ~ dmnorm(mu[1:p], omega.inv[1:p, 1:p])
        }
        tau ~ dgamma(tau.a, tau.b)
        sigma <- 1 / sqrt(tau)

        mu[1:p] ~ dmnorm(mu.prior.mean[1:p], mu.prior.precision[1:p, 1:p])

        omega.inv[1:p, 1:p] ~ dwish(omega.inv.matrix[1:p, 1:p], omega.inv.dof)
        omega[1:p,1:p] <- inverse(omega.inv[,])
}
```

Loop through the data, here using offset

Interindividual variability (IIV)

Inverse-Wishart prior for IIV

Interindividual variability

- Usually we can assume a Normal distribution which means log-normal when the parameters are on a log-scale
- This normal distribution is usually multivariate to account for correlations between parameters
- For multivariate inter-individual variability the Inverse-Wishart distribution is the only option for the variance-covariance matrix
- But partitioning is possible in groups with multivariate assumptions for relevant parameters, e.g. CL and V, and setting others independent
- Normal distribution may sometimes be replaced by a Student-t distribution which is more long tailed to make room for outliers

$$\boldsymbol{\mu} \sim \mathrm{St}(\mathbf{m}, \mathbf{P}^{-1}, n)$$

•Where *n* is a small integer. For a large *n* Student-t is identical to Normal

Wishart distribution

- The multivariate equivalent of chi-squared or gamma distribution
- Least informative when degrees of freedom = dimension
- It produces symmetric positive-definite matrices
- Not entirely non-informative
- But it is the only option for multivariate variance terms and unlike univariate variance terms uniform distributions cannot be used.
- In Winbugs the parameterisation of Wishart dist. differs from that of literature:

In most literature:	**$\mathrm{Wish}(\Sigma,\nu)$** where Σ scale matric and ν deg of freedom.
Such that:	**mean = $\nu\Sigma$**
But in Winbugs:	**dwish($R=\Sigma^{-1}$, ν)**
Such that:	**inv(mean) = $nR^{-1} = \Sigma/\nu$** so a reasonable choice for **$R=\nu\Omega$**

Data with hierarchical structure

Data with hierarchical structure, such measurements corresponding to different subjects and where this information needs to be preserved can be organised in 3 different ways:

1. Using an offset vector:

Model

```
for (i in 1:n.ind) {
    for (j in off.data[i]:(off.data[i + 1] - 1)) { …
```

Data

```
off.data=c(1, 3, 5, 7, 9, …

data=c(3.36365, 3.79197, 1.33584, 3.18724, 3.25417, 3.44967,
4.7714, 3.36978, 3.30257, 3.43646,…

time=c(1, 4, 0.08, 6, 2.5, 3, 2.5, 10, 0.25, 10, …
```

Data with hierarchical structure

2. Use nested indexing

Model

```
for (j in 1:n.obs) {
    data[j] ~ dnorm(model[j], tau)
    model[j] <- PK1abs1m(theta[ind.id[j],1:p],dose,time[j])
    }

for (i in 1:n.ind) {
    theta[i, 1:p] ~ dmnorm(mu[1:p], omega.inv[1:p, 1:p])
```

Data

```
ind.id=c(1, 1, 2, 2, 3, 3, 4, 4, 5, 5, …

data=c(3.36365, 3.79197, 1.33584, 3.18724, 3.25417, 3.44967,
4.7714, 3.36978, 3.30257, 3.43646,…

time=c(1, 4, 0.08, 6, 2.5, 3, 2.5, 10, 0.25, 10, …
```

Data with hierarchical structure

3. Use a matrix and put NA where there is no data

Model

```
for (i in 1:n.ind) {
    for (j in 1:8) { …
```

Data

```
time[]   data[,1]  data[,2]  data[,3]  data[,4]  data[,5]
  0.08         NA   1.33584        NA        NA        NA
  0.25         NA        NA        NA        NA   3.30257
     1    3.36365        NA        NA        NA        NA
   2.5         NA        NA   3.25417    4.7714        NA
     3         NA        NA   3.44967        NA        NA
     4    3.79197        NA        NA        NA        NA
     6         NA   3.18724        NA        NA        NA
    10         NA        NA        NA   3.36978   3.43646
```

Or use the S-Plus structure format

```
data=Structure(.Data=c(…),.Dim=c(…))
```

Descriptive covariates

- A covariate is an **independent variable** in the structural model.
- Usually there is at least one covariate in the model: **time**
- Additional covariates may be used to describe better the data
- Especially in population data, covariates can describe partly the observed variability
- Covariates may include: demographic data such as weight, age, sex, etc or even genetic data such as presence of certain alleles of genes
- Covariates are not parameters and are independent variables included in the dataset, like time.
- When covariates are included in the model additional parameters are also included to build the functional relationship of the covariate in the model

Mathematical formulation for covariates

Assuming a structural parameter of the model with some physical meaning
e.g. a volume of distribution in a compartment (V)

A covariate e.g. weight (WT) may be included as a linear function

$V = \theta_1 * WT$

Where θ_1 is a coefficient to be estimated, values for WT are provided in the data

Apart from a coefficient an intercept could be included

$V = \theta_1 + \theta_2 * WT$

It is often better to parameterise such that the covariate is centred

$V = \theta_1 + \theta_2 * (WT - WT_{mean})$

Then when $WT = WT_{mean}$, $V = \theta_1$

Mathematical formulation for covariates

Exponential relationships are also useful

$V = \theta_1 * \exp(\theta_2 * WT)$

Or centred

$V = \theta_1 * \exp(\theta_2 * (WT - WT_{mean}))$

These are linear on the log-scale

$\log V = \log \theta_1 + \theta_2 * (WT – WT_{mean})$

And power-laws

$V = \theta_1 * WT^{\theta 2}$ centred $V = \theta_1 * (WT / WT_{mean})^{\theta 2}$

Including covariates in WinBUGS

```
model {
        dose<-600000
        for (i in 1:n.ind) {
                for (j in off.data[i]:(off.data[i + 1] - 1)) {

                        data[j] ~ dnorm(model[j], tau)
                        model[j] <- PK1abs1m(theta[i,1:p],dose,time[j])
                }
                theta[i, 1:p] ~ dmnorm(theta.mean[1:p], omega.inv[1:p, 1:p])
                theta.mean[i, 1] <- mu[1]
                theta.mean[i, 2] <- mu[2] + mu[3]*(WT[i]-WT.mean)
                theta.mean[i, 3] <- mu[4]

        }
        tau ~ dgamma(tau.a, tau.b)
        sigma <- 1 / sqrt(tau)

        mu[1:q] ~ dmnorm(mu.prior.mean[1:q], mu.prior.precision[1:q, 1:q])

        omega.inv[1:p, 1:p] ~ dwish(omega.inv.matrix[1:p, 1:p], omega.inv.dof)
        omega[1:p,1:p] <- inverse(omega.inv[,])
}
```

Covariate assignment

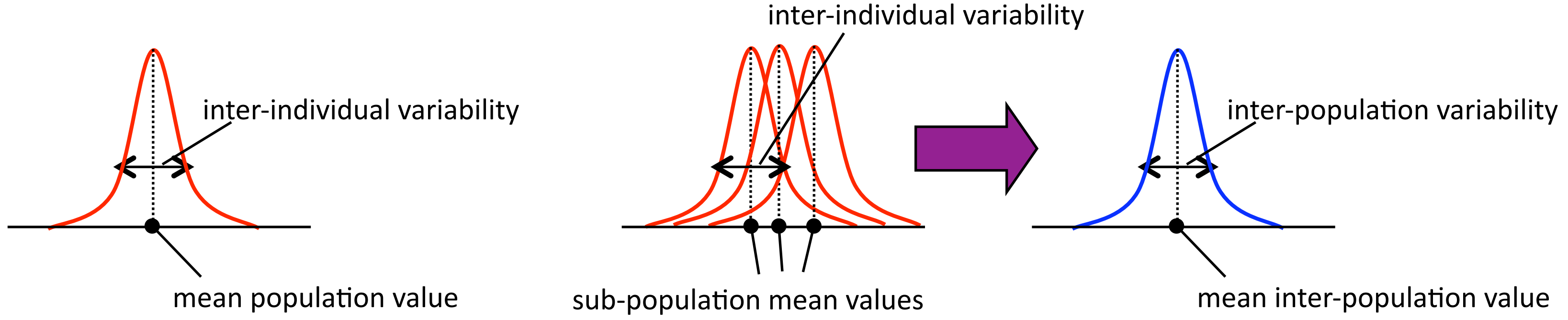
Additional hierarchical levels

- To include an inter-individual variability is the most common use of hierarchical modelling
- But more levels can be added to account, for example, inter-study or inter-lab variability, as studies performed in different labs may have unexplained variability
- Also an inter-occasion variability can account for unexplained variability between different occasions but in the same individual

Hierarchical model of 4 levels including inter-study variability (or inter-occasion)

3-level model

4-level model


inter-individual variability
mean population value
inter-individual variability
sub-population mean values
inter-population variability
mean inter-population value

structural model $Y_{kij}=f(\zeta_{ki},x_{ij})+\varepsilon_{kij}$

residual variability $\varepsilon_{kij} \sim N(0,\sigma_k)$

monitor

sub-population (or inter-occasion) $\zeta_{ki} \sim N(\boldsymbol{\theta}_k , \boldsymbol{\Omega})$

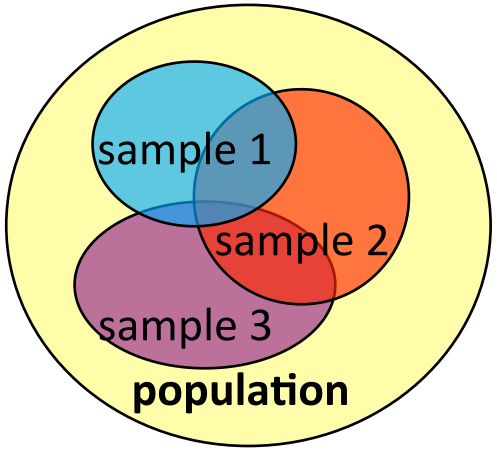
inter-population (or inter-individual) $\boldsymbol{\theta}_k \sim St(\boldsymbol{\mu}, \boldsymbol{\Sigma}, v)$ or $\boldsymbol{\theta}_k \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

uncertainty (prior) $\boldsymbol{\mu} \sim N(\mathbf{m},\mathbf{P}^{-1})$ $\boldsymbol{\Sigma}^{-1} \sim W(\mathbf{S}_1 ,v_1)$ $\boldsymbol{\Omega}^{-1} \sim W(\mathbf{S}_2 ,v_2)$ $\sigma_k^{-2} \sim \Gamma(a_k ,b_k)$

Prior influences only indirectly the parameters of interest

Exchangeability

Bayesian analysis is ideal for combining information from different sources.



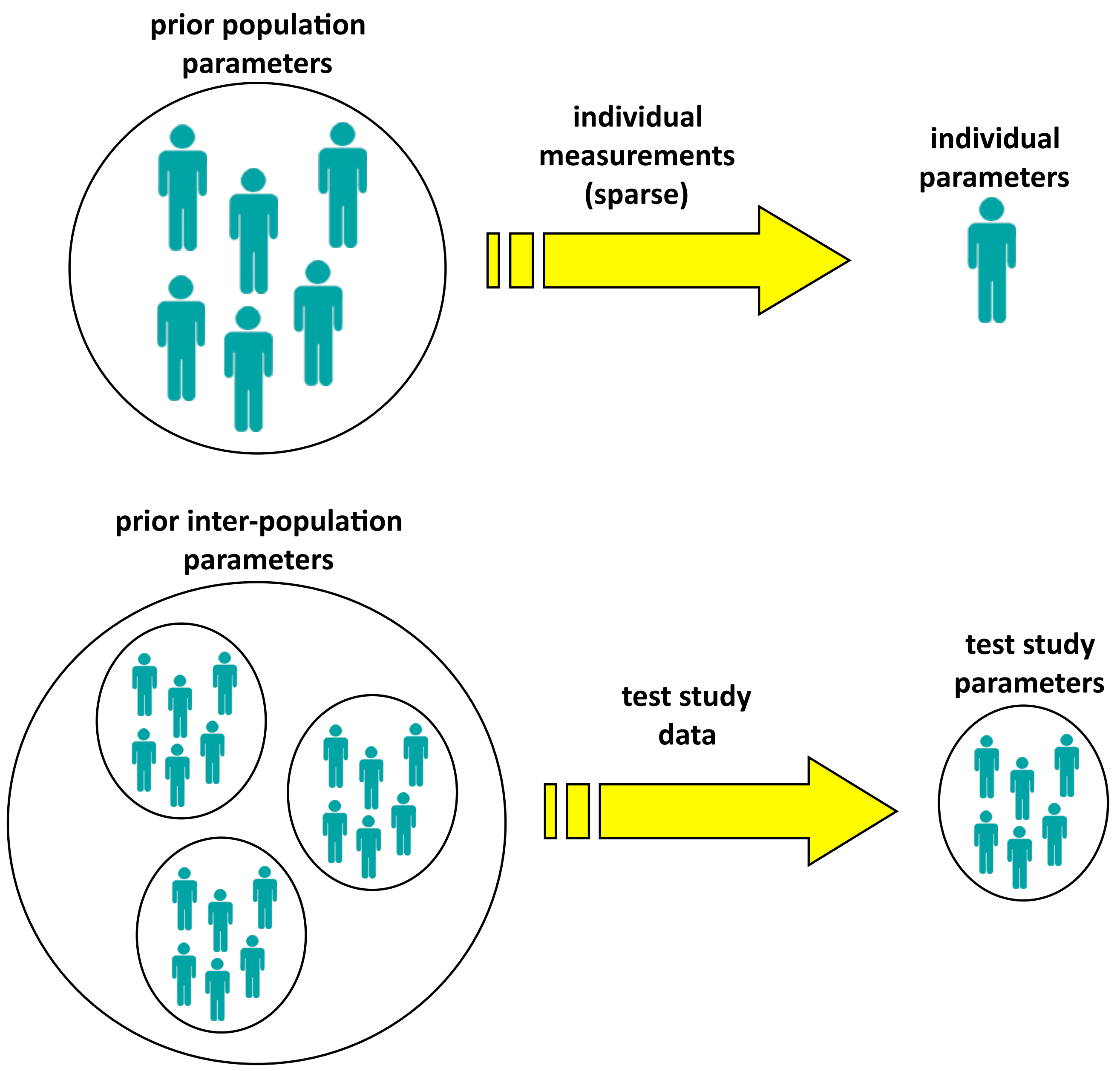
When combining datasets the populations are considered exchangeable

Override population non-exchangeability by modelling population differences such that they become exchangeable.

a) Systematic bias: model with covariates

b) Random differences: model with an extra level of hierarchy which accounts for the random differences.

Bayesian individualisation


prior population parameters
individual measurements (sparse)
individual parameters
prior inter-population parameters
test study data
test study parameters

Multilevel hierarchical data structures

When multiple hierararchical levels are present the data structures become more complicated and the use of nested indices or offsets is necessary

Example with 2D offset as a matrix

```
for (k in 1:n.st) {
    for (i in 1:n.ind) {
        for (j in (off.data[k,i]):(off.data[k,i + 1] - 1)) {
```

Nested indexing

```
for (j in 1:n.obs) {
    data[j] ~ dnorm(model[j], tau)
    model[j] <- Somemodel(zeta[st.id[j], ind.id[j],1:p],time[j])
    }
for (k in 1:n.st) {
    for (i in 1:n.ind) {
        zeta[k, i, 1:p] ~ dmnorm(theta[k,1:p], omega.inv[1:p, 1:p])
        }
    theta[k, 1:p] ~ dmnorm(mu[1:p], sigma.inv[1:p, 1:p])
```

Informative prior assignment

- The great advantage of a Bayesian analysis is the potential of using priors
- When non-informative priors are used a Bayesian analysis is a equivalent to a frequentist approach but the priors may not be completely non-informative
- Sources of prior information may be literature, or simply a previous run
- Typically prior information would be available in the form of expected values and associated uncertainty, but in WinBUGS we need to convert this into hyperparameters for parametric distributions (normal , gamma, Wishart, etc)
- A non-parametric discrete prior assignment is technically possible by using the following trick. However possibly slow and with numerical problems:

Prior for parameter D is one of a set of values, d[1], ..., d[K], with probabilities p[1], ..., p[K], then specify the arrays d[1:K] and p[1:K] and use:

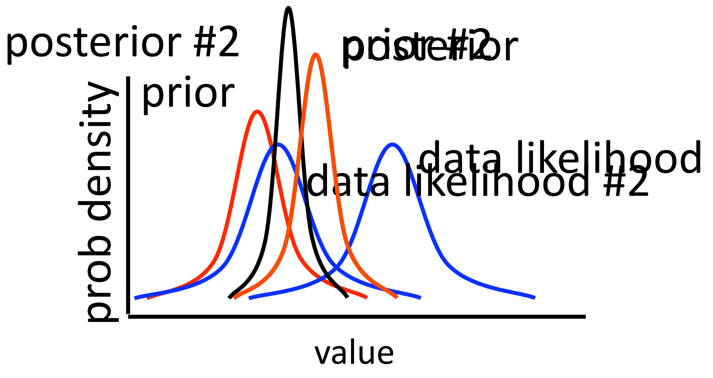
```
M ~ dcat(p[])
D <- d[M]
```

Summary - hierarchical

- Hierarchical models are implemented naturally within the Bayesian formulation
- Usually a 3 level approach including inter-individual variability is used
- It is possible to include covariates other than time
- More levels are possible including inter-occasion or inter-site variability
- Given a population estimates Bayesian individualisation is possible
- Importance of prior and data exchangeability

Sequential use of Bayes' theorem

The product of the prior and the data likelihood normalised gives the **posterior distribution**



$$P(\theta|\mathbf{X}) = \frac{p(\theta)\ l(\mathbf{X}|\theta)}{\int p(\theta) l(\mathbf{X}|\theta) d\theta}$$

So the result of a WinBUGS run can be used as a prior for a future run

Parameterise distributions (from WinBUGS runs)

To convert expected values and associated uncertainty (var) to hyperparameters of parametric distributions when these come from chains of a previous WinBUGS run, we can use formulas

Typically in analysis of kinetic data we are dealing with the following distributions

Normal: Hyperparameters are mean and precision (in WinBUGS) which are readily available

Gamma (for the inverse of a variance term): We make use of the fact that

$$\mathrm{E}(\tau_{\text{post}}) = a/b \qquad \mathrm{var}(\tau_{\text{post}}) = a/b^2$$

Which means that:

$$a = \frac{\mathrm{E}^2(\tau_{\text{post}})}{\mathrm{var}(\tau_{\text{post}})} \qquad b = \frac{\mathrm{E}(\tau_{\text{post}})}{\mathrm{var}(\tau_{\text{post}})}$$

Parameterise distributions (from WinBUGS runs)

Wishart (for the inverse of multivariate variance terms):
We make use of the fact that

$$\mathrm{E}(W) = \nu \cdot \mathbf{S} \qquad \mathrm{var}(W) = 2\nu \cdot \sigma_{ii}^2$$

Where σ_{ii} are diagonal elements of **S**

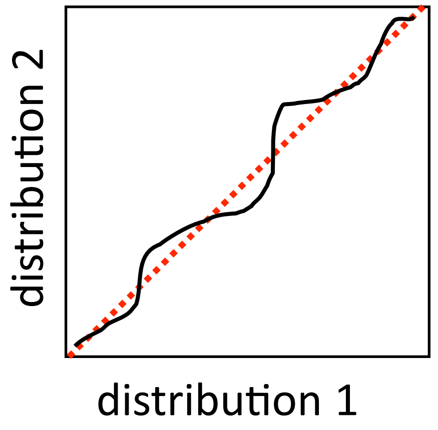
So we have:

$$\nu = \left[\frac{2 \cdot \mathrm{E}^2(\boldsymbol{\Omega}_{\text{post}}^{-1})_{ii}}{\mathrm{var}(\boldsymbol{\Omega}_{\text{post}}^{-1})_{ii}} \right] \qquad \mathbf{S} = \frac{\mathrm{E}(\boldsymbol{\Omega}_{\text{post}}^{-1})}{\nu}$$

Remember that:
parametrisation of Wishart in WinBUGS uses $R=S^{-1}$
Will obtain as many *ν* as the number of elements and have to choose the average or the minimum (least informative)

q-q plots (diagnostic plot)

- Plot the quintiles of 2 distributions
- If the graph is on the identity line (…) the distributions are the same



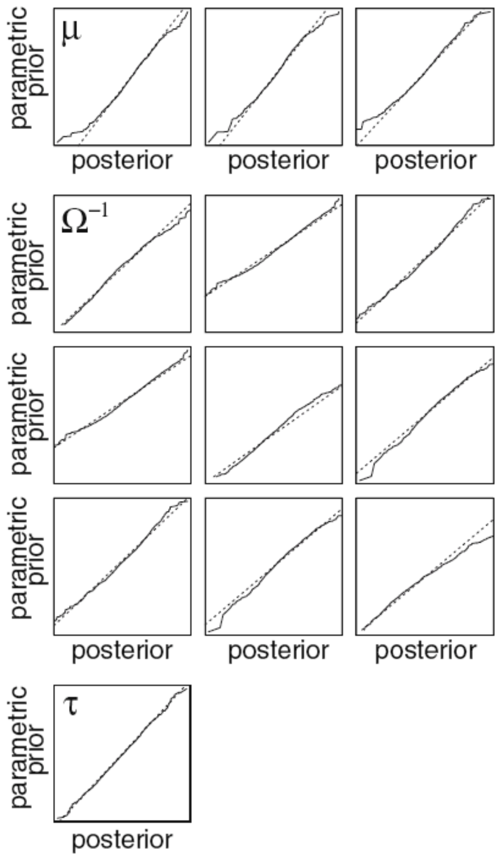
Issues from combining information

Example: Usual model 1-compartment PK 1st order absorption model
2 datasets same number of subjects, both datasets from the same population:

We want to do sequential analysis and compare it to combined analysis

Fit the first dataset with WinBUGS and use the results as priors to fit the second dataset

First parametrise the result of the first run using the formulas: looks ok



Normal (population means)

$$\mathbf{m} = (7.15719, 8.92433, -0.269377), \quad \mathbf{P} = \begin{pmatrix} 81.34 & -40.12 & -9.477 \\ -40.12 & 72.3863 & -15.61 \\ -9.477 & -15.61 & 25.16 \end{pmatrix}$$

Wishart (IIV)

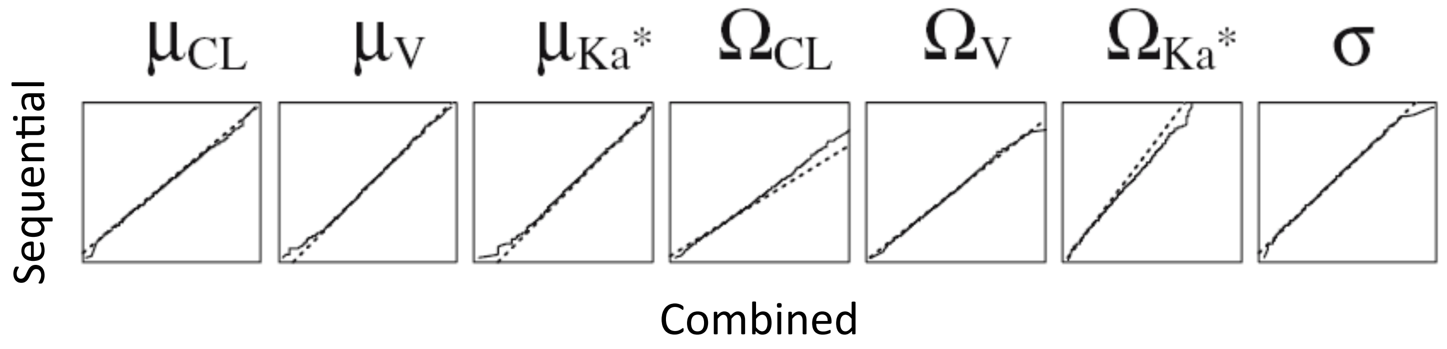
$$\nu = 8, \quad \mathbf{S} = \begin{pmatrix} 1.383 & 1.063 & 1.168 \\ 1.063 & 1.614 & 1.007 \\ 1.168 & 1.007 & 2.707 \end{pmatrix}$$

Gamma (Res error)

$$a = 84.17, \quad b = 3.939$$

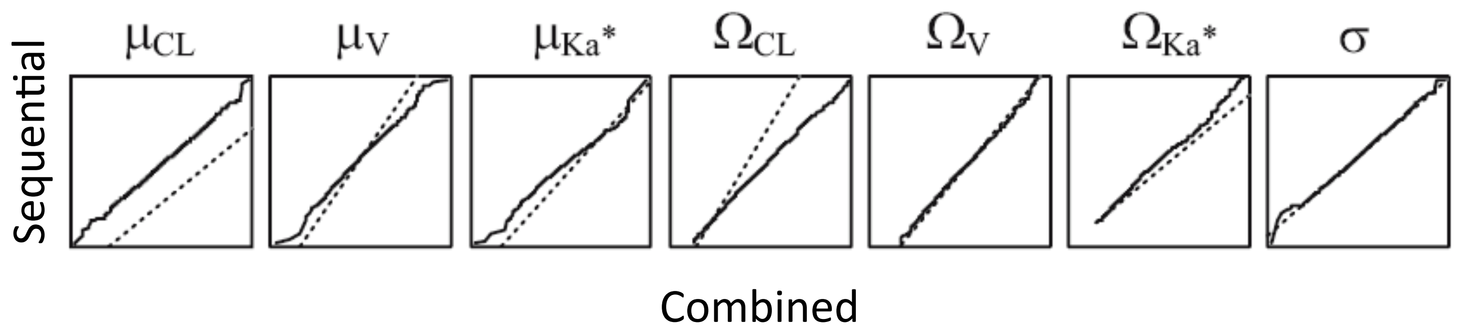
Issues from combining information

Then use this parametric prior to analyse the second dataset and compare to combined analysis: (**GOOD!**)



But the datasets were from the same population what if they weren't?

Same problem but one parameter different between the 2 datasets (**NOT GOOD!**):



Issues from combining information

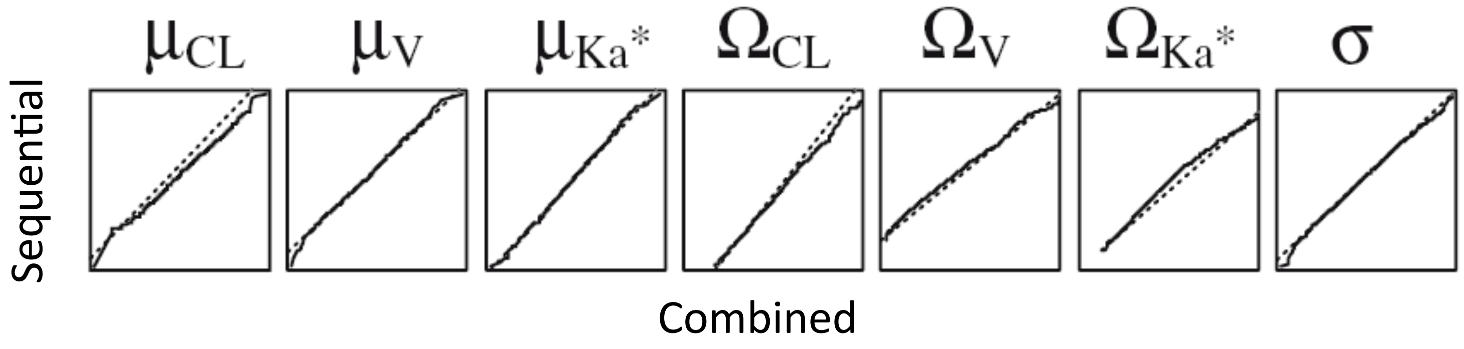
The reason is that the independent Normal and Inverse-Wishart distributions are not the conjugate prior for this problem. It is called semi-conjugate and is used for convenience.

$$\boldsymbol{\mu} \sim \mathrm{N}(\mathbf{m},\mathbf{P}^{-1}) \qquad \boldsymbol{\Omega} \sim \text{Inv-Wish}(\boldsymbol{\Sigma},v)$$

The true conjugate is this:

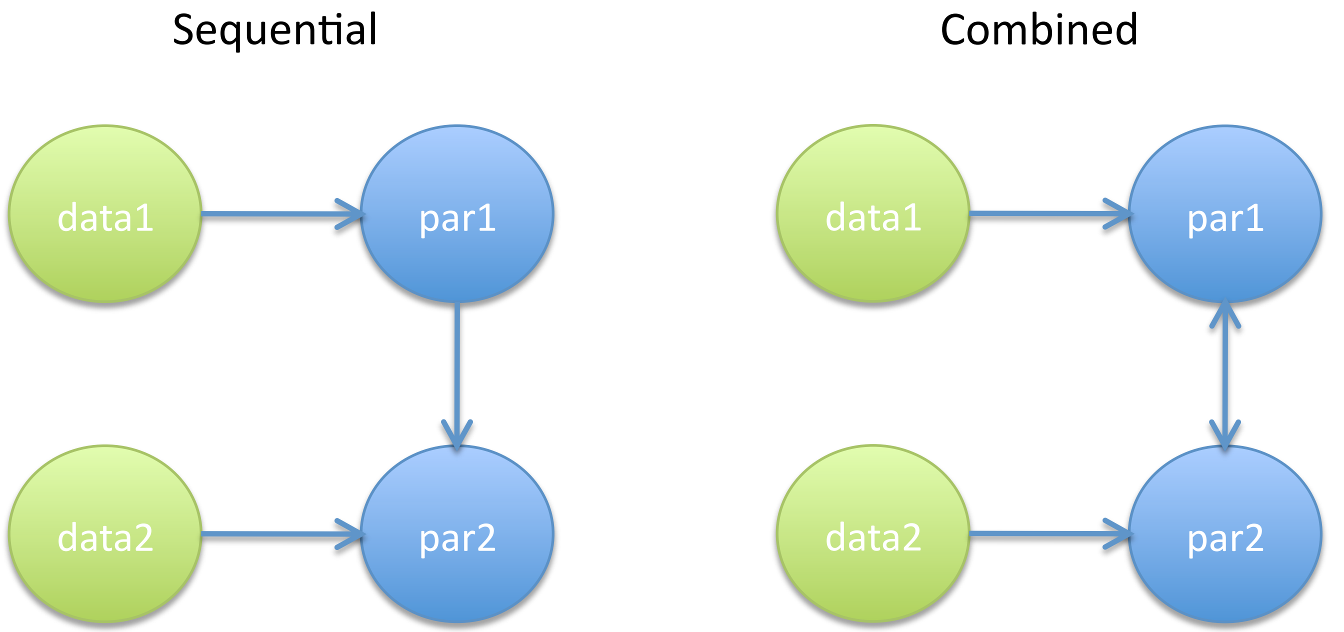
$$\boldsymbol{\mu} \sim \mathrm{N}(\mathbf{m},\boldsymbol{\Omega}/\mathrm{c}) \qquad \boldsymbol{\Omega} \sim \text{Inv-Wish}(\boldsymbol{\Sigma},v)$$

Where *v* is degrees of freedom. This is the Normal-Inverse-Wishart (NIW) distribution. It is not often used because it is too restrictive but when applied in the problem we saw before (**FIXED!**):



The “cut” function

We looked at sequential vs combined analysis and it compared well but there is a difference between the 2 in terms of information flow



The “cut” function can work as a “valve” of information flow, preventing the updating of the parameter being cut, but allowing it to contribute to the rest of the model

```
par.cut <- cut(par)
```

Parameterise distributions (from literature)

In literature inverse variance terms will probably not be available but instead the variance terms themselves would be reported (if you are lucky)

This is not trivial because the average of inverses is not equal to the inverse of the average

Therefore we can use the following:

Inverse-Gamma:

$$\hat{\sigma}^2 = \frac{b_R}{a_R - 1} \qquad \mathrm{var}(\sigma^2) = \frac{b_R^2}{(a_R-1)^2(a_R-2)}$$

From which it follows that:

$$a_R = \frac{(\hat{\sigma}^2)^2 + 2\mathrm{var}(\sigma^2)}{\mathrm{var}(\sigma^2)} \qquad b_R = \frac{(\hat{\sigma}^2)^3 + \hat{\sigma}^2\mathrm{var}(\sigma^2)}{\mathrm{var}(\sigma^2)}$$

Parameterise distributions (from literature)

Inverse-Wishart:

$$\widehat{\Omega}_{kk} = \frac{\Sigma^{kk}}{v - p - 1} \qquad \operatorname{var}(\Omega_{kk}) = \frac{2(\Sigma^{kk})^2}{(v - p - 3)(v - p - 1)^2}$$

From which it follows that:

$$v_k = \frac{(3 + p)\operatorname{var}(\Omega_{kk}) + 2(\widehat{\Omega}_{kk})^2}{\operatorname{var}(\Omega_{kk})} \qquad \boldsymbol{\Sigma} = \frac{(\widehat{\boldsymbol{\Omega}})^{-1}}{v - p - 1}$$

Note that:
parametrisation of Wishart in WinBUGS uses $R=\Sigma^{-1}$
Will obtain as many v_k as the number of elements and have to choose the average or the minimum (least informative) and set it as your final v

Parameterise distributions (from literature)

But also for the populations means, we saw that the NIW prior may be preferable to the simpler but more flexible partitioned prior.

In the case we want to use the NIW, since the covariance of the Normal distribution is :

$$\operatorname{cov}(\boldsymbol{\mu}) = \widehat{\boldsymbol{\Omega}}/c$$

We have to calculate the degrees of freedom c as an average:

$$c = \operatorname{Tr}\big[\widehat{\boldsymbol{\Omega}} \cdot (\operatorname{cov}(\boldsymbol{\mu}))^{-1}\big]/p$$

Notes on using informative priors from previous studies

- Applying a prior directly on a parameter of interest is a strong assumption. The result is a weighted average between the prior and the data likelihood

- Exchangeability of prior and dataset is important in this case because of the very drastic influence of the prior

- Applying a prior on a hyperparameter which sits above the parameter of interest in a hierarchical structure is a more conservative assumption.

- NIW prior may be preferable to the semi-conjugate prior but more restrictive

Summary - priors

- Sequential applications of Bayes' theorem: output of WinBugs may be used as prior
- Output of WinBUGS is non-parametric while priors need to be parametric
- One way to parameterise is by using formulas of mean and variance for the different distributions
- Mentioned versions of these appropriate for WinBUGS output and others appropriate for estimates reported in literature

Model selection

- Between 2 competing models which one best describes the data?
- This has to take into account the model complexity as models with more parameters may have a better fit but the improvement may not be significant
- Routinely, Akaike Information Criterion (AIC) may be used:

$$AIC = 2k - 2\log L$$

- Where k is the number of parameters and L is the maximum likelihood value.
- The model with the lower AIC is the preferable
- The model closer to the data will have the smallest “-2 log L” term and the model with the highest number of parameters is penalised for that fact

Deviance Information Criterion (DIC)

A generalisation of AIC is Deviance Information Criterion (DIC)*

$$DIC = p_D + \overline{D}$$

where D bar is the posterior mean of the **deviance** a measure of goodness of fit

Deviance = -2 Log L(data|θ)

p_D is the effective number of parameters and calculated as mean posterior deviance (D bar) minus deviance at posterior mean of the parameters

$$\begin{aligned} p_D &= E_{\theta|y}[D] - D(E_{\theta|y}[\theta]) \\ &= \overline{D} - D(\overline{\theta}); \end{aligned}$$

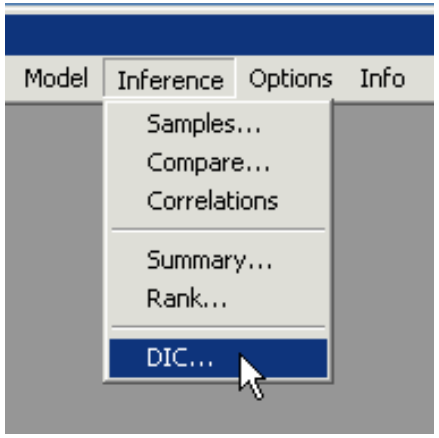
In non hierarchical models p_D is approximately equal to the actual number of parameters. In general it isn't.

* Spiegelhalter D. et al., Journal of the Royal Statistical Society, Series B, 2002 64(4):583-616.

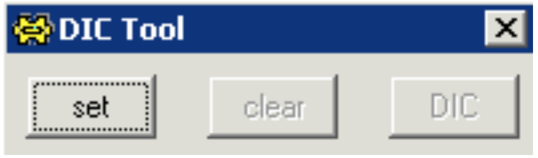
DIC in WinBUGS

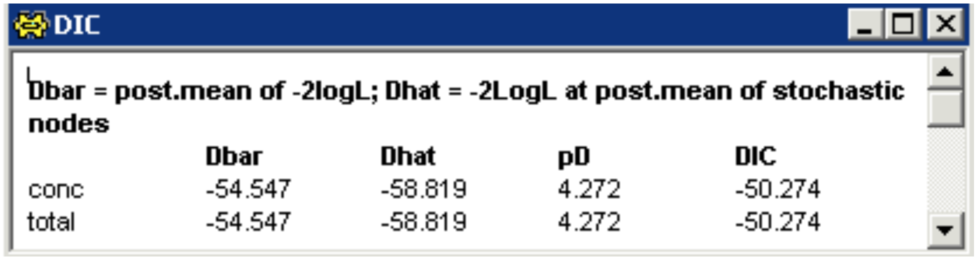
Because of its nature DIC can be readily calculated in an MCMC run

Conveniently, it is readily available as an item in WinBUGS "Inference" menu



This opens a dialog box which allows monitoring it and provides detailed output for DIC





```
Dbar = post.mean of -2logL; Dhat = -2LogL at post.mean of stochastic
nodes
        Dbar       Dhat       pD       DIC
conc    -54.547    -58.819    4.272    -50.274
total   -54.547    -58.819    4.272    -50.274
```

Characteristics of DIC

- Readily available in WinBUGS
- Akaike like behaviour: lowest DIC better model and penalises additional parameters
- DICs are comparable only over models with exactly the same observed data, but there is no need for them to be nested
- Differences in DIC values considered significant (Very roughly):
  - differences of more than 10, are considered significant.
  - differences between 5 and 10 are substantial
  - differences in DIC less than 5, and the models make very different inferences, then it could be misleading just to report the model with the lowest DIC
- Value of p_D is dependant on parameterisation (disadvantage) and may even be negative

Other ways for Bayesian model selection

- Another version of DIC which uses quantity p_V instead of p_D has been implemented in R2Winbugs, where p_V is invariant to parameterisation

- Bayesian Information Criterion (BIC) is an alternative to DIC but is not straightforward to calculate in WinBUGS

- Bayes Factors can also be used for model selection in the way a likelihood - ratio test is used

- **But:** DIC – BIC – Bayes Factors, have different aims and are not really alternatives for the same problem