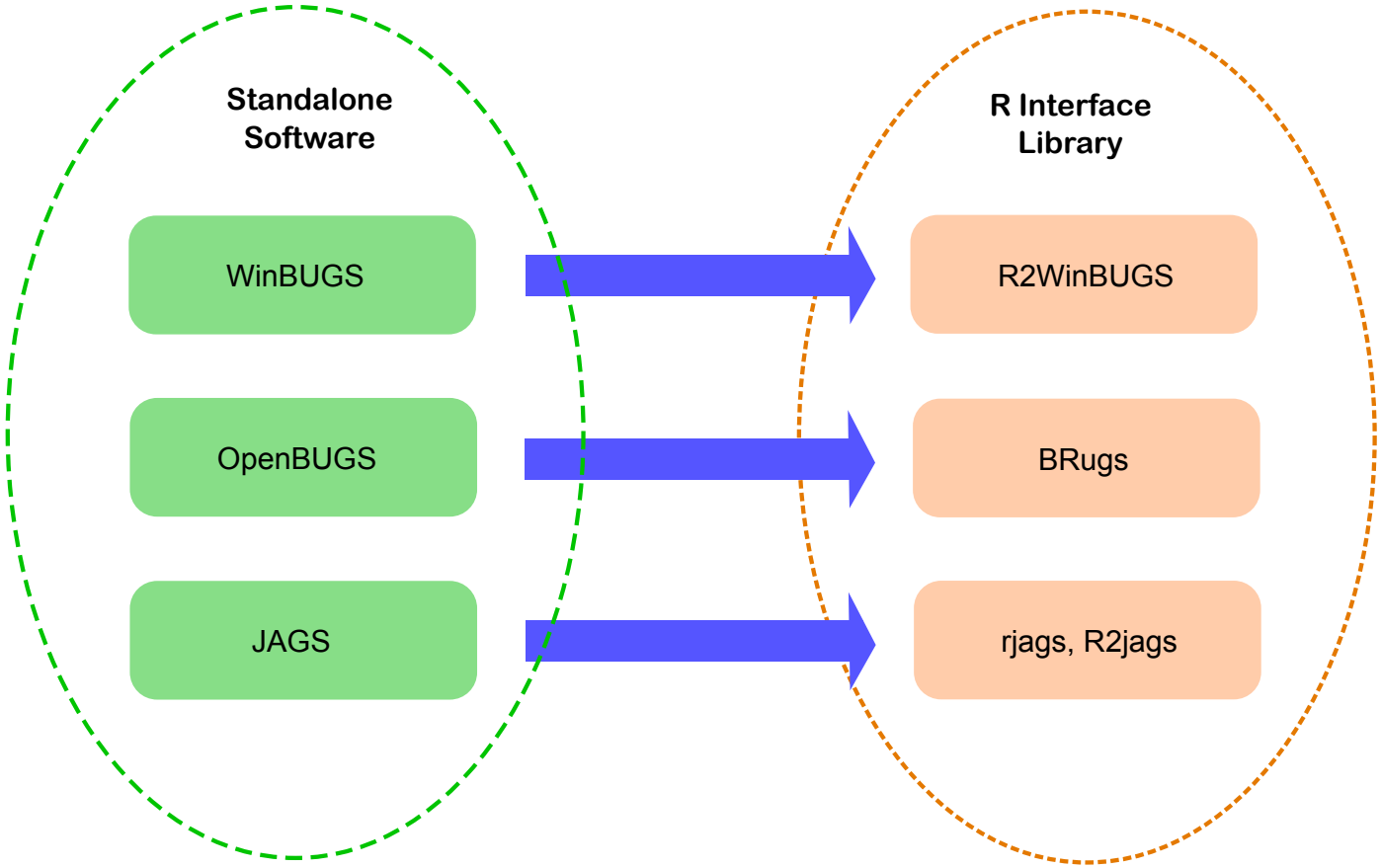
R Interface for BUGS and Stan

Lin Zhang

Department of Biostatistics
School of Public Health
University of Minnesota

Overview



Pros: user friendly GUI
Cons: cannot read data from file; tricky to do posterior inference

Cons: No interactive GUI
Pros: R interface facilitates reading data from file, posterior processing, plotting, etc.

Relevant Softwares

To use the R implemented BUGS, we need to download and install the standalone softwares as well as the install the R packages in R:

- R2WinBUGS:
  - calls **WinBUGS** from within R
  - works in windows OS
- BRugs:
  - calls **OpenBUGS** from within R
  - information about OpenBUGS: `http://www.openbus.info/w`
  - runs in Windows, Mac, and Linux (not efficient in Linux)
- rjags:
  - calls **JAGS** from within R
  - available at `http://mcmc-jags.sourceforge.net`
  - runs stably in Windows, Mac, and Linux

Toy Example

- Suppose the simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

- Thus the likelihood is:

$$y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$$

- Assign vague prior:

$$\begin{aligned} \pi(\beta_0) &= 1 \\ \pi(\beta_1) &= 1 \\ \pi(\sigma) &= Unif(0.01, 100) \end{aligned}$$

WinBUGS code

```
model{
    for(i in 1:N) {
        y[i] ~ dnorm(mu[i],y_prec)
        mu[i] <- beta[1]+beta[2]*x[i]
    }
    beta[1] ~ dnorm(beta_mean[1],beta_prec[1])
    beta[2] ~ dnorm(beta_mean[2],beta_prec[2])
    y_prec <- 1/(sigma*sigma)
    sigma ~ dunif(0.01,100)
}
# Data
list(x = c( 0.0, 0.41, 0.41, ...),
     y = c(1.80, 1.85, 1.87, ...),
     N = 27,
     beta_mean = rep(0,2), beta_prec = rep(0.01,2))
# Inits
list(beta=c(1,1),sigma=1)
list(beta=c(0,0),sigma=0.5)
```

Preparation: write model to .txt file

- Before running BUGS in R, first we need to write and save the Bayesian model in a .txt file.
- Three ways to do it:
  - Use the BRugs/R2WinBUGS function **'writeModel'**
  - Use the base R command **'writeLines'**
  - Directly write in a text editor
- Code ...

BRugs: model built step by step

- Recall the steps in WinBUGS:
  1. load model
  2. load data
  3. compile with specified number of chains
  4. load initials
  5. 'update' for burn-in iterations
  6. set parameters of which posterior samples will be collected
  7. 'update' again to collect posterior samples
  8. check results
- BRugs in R has the corresponding command for each step
- Code ...

BRugs: meta function

- BRugs has a meta function **'BRugsFit'** to run model in one step

```
>BRugs.res <- BRugsFit(modelFile=modelfile,
      data=list(x=x,y=y,N=N,beta_mean=beta_mean,
      beta_prec=beta_prec), inits=initList,
      numChains=2, parametersToSave = c("beta","sigma"),
      nBurnin=1000, nIter=10000, nThin=1,
      DIC=TRUE )
```

- Summary statistics and DIC values are returned

```
>BRugs.res$Stats
>BRugs.res$DIC
```

- You can also use the same functions to check results, extract posterior samples

```
>samplesStats("*")
>samplesHistory("*")
```

- Code ...

CODA samples

- Set the argument 'coda=TRUE', the function 'BRugsFit' will return posterior samples in coda format.

```
>codaSamples <- BRugsFit(modelFile=modelfile,
        data=list(x=x,y=y,N=N,beta_mean=beta_mean,
        beta_prec=beta_prec), inits=initList,
        numChains=2, parametersToSave = c("beta","sigma"),
        nBurnin=1000, nIter=10000, nThin=1,
        coda=TRUE )
```

- The resulting 'codaSamples' has these indices:

```
>codaSamples[[chainIdx]][iteratIdx, paramIdx]
```

  Posterior samples of certain parameter can be extracted and saved for future use

- We can use the 'CODA' R package for MCMC diagnostics.

Convergence diagnostics with 'CODA' package

- Trace plots

```
>traceplot(codaSamples)
>traceplot(codaSamples[,'beta[1]')
```

- Gelman-Rubin diagnostic

```
>gelman.diag(codaSamples)
>gelman.plot(codaSamples)
```

- Autocorrelation plots

```
>autocorr.plot(codaSamples)
```

- Cross-correlation

```
>crosscorr(codaSamples)
```

- Effective sample size

```
>effectiveSize(codaSamples)
```

rjags

- Three steps for Bayesian Analysis
  1. Create, initialize, and adapt the model:

```
>jagsModel <- jags.model(file=modelfile,
          data=list(x=x,y=y,N=N,beta_mean=beta_mean,
          beta_prec=beta_prec), inits=initList,
          n.chains = 2, n.adapt = 1000 )
```

  2. Burnin interations with 'update' function

```
>update(jagsModel, n.iter=1000)
```

  3. Collect posterior samples in 'coda format'

```
>codaSamples <- coda.samples(model=jagsModel,variable.names
            = c("beta","sigma"), n.iter = 10000, thin = 1)
```

- Note the 'rjags' package automatically load 'coda' package. Thus 'coda' functions can be directly used for convergence diagnostics for outputs from 'coda.samples' function.
- DIC calculation in rjags:

```
>dic.samples(model=jagsModel,n.iter=10000,thin=1,type='pD')
```

- Code ...

R2WinBUGS

- Similar to the meta function in package 'BRugs'
- Main function

```
>bugs.res <- bugs(data=list(x=x,y=y,N=N,beta_mean=beta_mean,
        beta_prec=beta_prec), inits=initList,
        parameters.to.save = c("beta","sigma"),
        model.file=modelfile,
        n.chains=2, n.iter=10000, n.burnin=1000, n.thin=1,
        DIC = TRUE, # codaPkg=TRUE,
        bugs.directory='C:/Program Files/WinBUGS14/')
```

- Check result

```
>print(bugs.res)
>plot(bugs.res)
```

- Return coda-format outputs if set the argument 'codaPkg=TRUE'.
- Code ...

Stan: Efficient MCMC for Bayesian Analysis

- Stan is a C++ library for Bayesian modeling and inference.
- Like BUGS and JAGS, Stan generates posterior samples given a user-specified hierarchical model and data $\rightarrow$ **No need to derive the posteriors and code up the MCMC algorithm.**
- Differently, Stan uses Hamiltonian Monte Carlo (HMC) and No-U-Turn Sampler (NUTS) to produce high dimensional proposals that are **accepted with high probability** without having to spend time tuning.
- Many interfaces: RStan (R), PyStan (Python), MatlabStan (MATLAB), CmdStan (shell, command-line terminal), ...

RStan: R interface to Stan

- The **rstan** package allows for fitting Stan models from R
- RStan has inbuilt diagnostics to analyse the MCMC outputs
- Typical workflow:
  1. Write a hierarchical model using Stan language and save in a file with `.stan` extension
  2. Run the Stan program by a single call to the **stan** function in R
  3. Diagnose non-convergence of the MCMC chains
  4. Conduct inferences based on the posterior samples

Simple Example: Eight Schools

- Data:

School	y	σ
A	28	15
B	8	10
C	-3	16
D	7	11
E	-1	9
F	1	11
G	18	10
H	12	18

- Random effects model:

$$\begin{aligned} y_j &\sim N(\theta_j, \sigma_j^2) \quad j = 1, \ldots, 8 \\ \theta_j &\sim N(\mu, \tau^2) \end{aligned}$$

- Stan and R codes ...

Stan codes

```
data {
    int<lower=0> J;           // number of schools
    real y[J];                // estimated treatment effects
    real<lower=0> sigma[J];   // s.e. of effect estimates
}
parameters {
    real mu;
    real<lower=0> tau;
    vector[J] eta;
}
transformed parameters {
    vector[J] theta;
    theta = mu + tau * eta;
}
model {
    eta ~ normal(0,1);
    y ~ normal(theta,sigma);
}
//model {
//    target += normal_lpdf(eta | 0, 1);
//    target += normal_lpdf(y | theta, sigma);
//}
```

R codes

```
> schools_data <- list(
+     J = 8,
+     y = c(28,  8, -3,  7, -1,  1, 18, 12),
+     sigma = c(15, 10, 16, 11,  9, 11, 10, 18)
+ )

> library(rstan)

> fit1 <- stan(
+      file = "schools.stan",  # Stan program
+      data = schools_data,    # named list of data
+      chains = 2,             # number of Markov chains
+      warmup = 1000,          # number of warmup iterations per chain
+      iter = 2000,            # total number of iterations per chain
+      cores = 2,              # number of cores (could use one per chai
+      refresh = 0             # no progress shown
+ )
```

Convergence diagnosis and posterior summary

- Check the convergence and summarize the posterior distribution

```
> traceplot(fit1, pars = c("mu", "tau"), inc_warmup = TRUE)
> print(fit1, pars=c("theta", "mu", "tau", "lp__"),
+       probs=c(.025,.5,.975))
```

- Check the between-parameter cross-correlation

```
> pairs(fit1, pars = c("mu", "tau", "lp__"), las = 1)
```

- Extract the posterior samples

```
> para_samples <- extract(fit1, pars=c("theta", "mu", "tau",
+                                      "lp__"), inc_warmup=FALSE)
```

- Some diagnostic tools for Hamilton Monte Carlo algorithm

```
> sampler_params <- get_sampler_params(fit1, inc_warmup = TRUE)
> summary(do.call(rbind, sampler_params), digits = 2)
```