BUGS language

- **Collect several definitions by using indexing and looping:**
  - for(i in 1:K){ X[i] ~ dbin(theta[i],N[i]) }
  - Give K as fixed value in the data listing.
  - Every definition within ”for(i in 1:K){ }” should have index i.
  - 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]”
  - Be careful to index correctly!

BUGS language

- **What distributions and logical functions are available?**
  - Check the list from manual/menu.
  - Pay attention to parameterization!
  - A very useful deterministic function: step()
  - What you define should be logically correct and computable in all situations. 1/X should not become 1/0 with stochastic node X.

BUGS language

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

```
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 data can be declared within model code

```
yy <- log(y)
yy ~ dnorm(mu,tau)
```

Here y would be given in data.

- Can check your data values from ’info’ → ’node info’ → ’values’

Some analyses with BUGS

- Diagnostic testing.
- Estimate a proportion.
- Compare proportions in two populations.
- Estimate a mean.
- Compare means in two populations.
- Linear & other regression.

Some analyses with BUGS

- **Diagnostic testing with additional data on sensitivity and specificity**

```
model{
x[1] ~ dbin(q[1],N[1])
x[2] ~ dbin(q[2],N[2])
y ~ dbin(pr,M); pr <- p*q[1]+(1-p)*(1-q[2])
q[1] ~ dunif(0,1); q[2] ~ dunif(0,1);
p ~ dunif(0,1)
}
list(x=c(45,28),N=c(50,30),M=100,y=10)
```

Some analyses with BUGS

- **Estimate and compare proportions**

```
model{
x ~  dbin(p,Nx);  p ~ dunif(0,1)
y ~  dbin(q,Ny);  q ~ dunif(0,1)
diff <-  p - q  # could assess the difference
r <- p/q  # could assess the ratio
pr <- step(diff)   # an indicator variable
}
list(x=3,y=7,Nx=20,Ny=45)
```

Some analyses with BUGS

- **Estimate and compare means**

```
model{
 for(i in 1:N){
 ahonen[i] ~  dnorm(m[1],tau[1])
 janda[i] ~  dnorm(m[2],tau[2])
 }
 for(i in 1:2){
 m[i] ~ dnorm(0,0.0001)
 tau[i] ~  dgamma(0.01,0.01)
 }
 diff <-  m[1]-m[2]
 pr <- step(diff)
 }
```

Some analyses with BUGS

- **Data:** scores of J Ahonen and J Janda from 8 ski jumping competitions (Four Hills tournament, 2006)

  list(ahonen = c(299.7, 255.2, 281.7, 238.0, 270.9, 262.2, 255.4, 293.0),
  janda = c(238.7, 285.6, 287.1, 252.2, 262.6, 264.7, 263.2, 291.0), N=8)

- Tips: with very flat, vague priors, BUGS may generate bad starting values (better to assign inits yourself).
- Generate predictions by adding this line:

  pred.ahonen ~ dnorm(m[1],tau[1])

Some analyses with BUGS

- **In 2006, they got exactly the same total score!**
- **Assume you have the results after 7 competitions. Make a prediction for the total score. What's the probability that the difference is < 1 point?**
- **Set the 8th result as 'NA' in data, then run the following:**

```
model{
    for(i in 1:N){
    ahonen[i] ~  dnorm(m[1],tau[1])
    janda[i] ~  dnorm(m[2],tau[2])
    }
    for(i in 1:2){
    m[i] ~ dnorm(0,0.0001)
    tau[i] ~  dgamma(0.01,0.01)
    }
    ahonen.total  <- sum(ahonen[1:N])
    janda.total  <- sum(janda[1:N])
      pr <-   1- step(abs(ahonen.total-janda.total)-1)
    }
```

Some analyses with BUGS

- **Linear regression**
  - **York rainfall data: x= in November, y= in December**

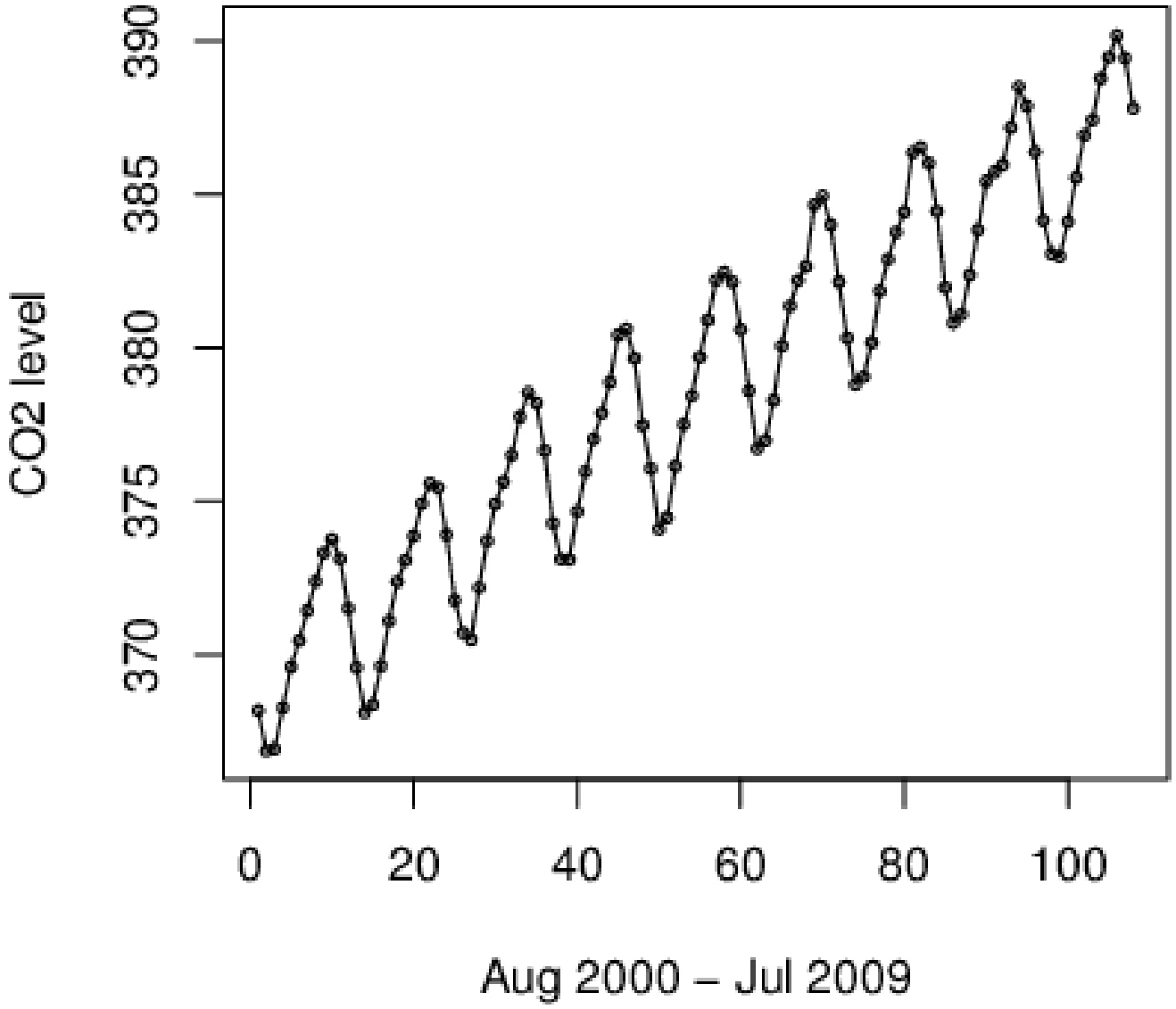
```
model{                          List(y = c(41,52,18.7,55,40,29.2,51,17.6,46.6,57),
for(i in 1:10){                 x = c(23.9,43.3,36.3,40.6,57,52.5,46.1,142,112.6,23.7))
y[i] ~ dnorm(mu[i],tau)
mu[i] <- beta[1] + beta[2]*x[i]
# mu[i] <- beta[1]+ beta[2]*(x[i]-mean(x[]))    # standardized covariates

for(i in 1:2){
beta[i] ~ dnorm(0,0.001)
}
tau ~ dgamma(0.01,0.01)
# prediction with given value xnew:
ynew ~ dnorm(munew,tau); munew <- beta[1] + beta[2]*xnew
# munew <-beta[1] + beta[2]*(xnew-mean(x[])) # standardized covariates
}
```

Interpretation of beta[1] in both cases? E(y | x=0) vs E(y|x=mean(x))

Some analyses with BUGS

- **Linear & nonlinear regression**
  - **Atmospheric CO_2, monthly, Mauna Loa, Hawaii**

list(N=120,x=c(368.18,366.87,366.94,368.27,369.62,370.47,
371.44,372.39,373.32,373.77,373.13,371.51,369.59,368.12,
368.38,369.64,371.11,372.38,373.08,373.87,374.93,375.58,
375.44,373.91,371.77,370.72,370.5,372.19,373.71,374.92,
375.63,376.51,377.75,378.54,378.21,376.65,374.28,373.12,
373.1,374.67,375.97,377.03,377.87,378.88,380.42,380.62,
379.66,377.48,376.07,374.1,374.47,376.15,377.51,378.43,
379.7,380.91,382.2,382.45,382.14,380.6,378.6,376.72,
376.98,378.29,380.07,381.36,382.19,382.65,384.65,
384.94,384.01,382.15,380.33,378.81,379.06,380.17,
381.85,382.88,383.77,384.42,386.36,386.53,386.01,
384.45,381.96,342,
385.72,385.96,387.18,388.5,387.88,386.38,384.15,
383.07,382.98,384.11,385.54,386.93,387.42,388.77,
389.46,390.18,389.43,387.81)


CO2 level
370
375
380
385
390
0
20
40
60
80
100
Aug 2000 – Jul 2009

Some analyses with BUGS

- **Linear and nonlinear terms**

```
model{
tau ~ dgamma(0.01,0.01);
for(i in 1:5){a[i] ~ dnorm(0,0.001)}
for(i in 1:N){
month[i] <- i
x[i] ~ dnorm(mu[i],tau)
mu[i]<- a[1]+a[2]*i+a[3]*sin(2*pi*i/12)+a[4]*cos(2*pi*i/12)
}
pi <- 3.1415926
}
```

Some analyses with BUGS

- **Generalized linear model: Poisson**
  - **Number of lung cancer cases**
  - **Population counts**
  - **In age groups, in different cities, in 1968-1971.**
  - **Use the first age group in the first city as a reference, to compute age effects and city effects**
  - **$\log(\lambda_{age,city}) = \mu_0 + \alpha_{age} + \beta_{city}$, with $\alpha_{age=1} = \beta_{city=1} = 0$**
  - **$X_{age,city} \sim \text{Poisson}(4\lambda_{age,city}\, pop_{age,city})$**

cases[] pop[] age[] city[]
11 3059 1 1
11 800 2 1
11 710 3 1
10 581 4 1
11 509 5 1
10 605 6 1
13 2879 1 2
6 1083 2 2
15 923 3 2
10 834 4 2
12 634 5 2
2 782 6 2
4 3142 1 3
8 1050 2 3
7 895 3 3
11 702 4 3
9 535 5 3
12 659 6 3
5 2520 1 4
7 878 2 4
10 839 3 4
14 631 4 4
8 539 5 4
7 619 6 4
END

Some analyses with BUGS

```
model{  # design matrix X could also be written beforehand in data
            # but it is here constructed from 'age' and 'city'.
            # The linear predictor can then be computed using inprod.
for(i in 1:24){
cases[i] ~ dpois(mu[i]); group[i] <- i
mu[i] <- pop[i]*4*lambda[i] # lambda = incidence per year
LA[i] <- lambda[i]/100000 # LA = inc. per 10^5 per year
log(lambda[i]) <- inprod(alpha[],X[i,])
X[i,1] <- 1
for(k in 2:6){X[i,k] <- equals(age[i],k) }
for(k in 2:4){X[i,k+5] <- equals(city[i],k) }
}
for(k in 1:9){  alpha[k] ~ dnorm(0,0.001)
A[k] <- exp(alpha[k])  }
}
```

Tips

- **Always think it as a DAG.**
- **Data variable has to correspond to a (fixed) 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 how slow or fast it is.**
- **Give constants in data, not within code.**
- **Separate clearly what’s data, what’s model.**
- **Use comments # there are never too many!**
- **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)**
- **Constants cannot be monitored, but can check them from node-info menu button.**
- **Sooner or later, it will be more convenient to run BUGS from R, try it.**
- **For the inbuilt convergence diagnostics, you should pick overdispersed starting values for at least 3 chains.**
- **Think of identifiability: is 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]+… … .**