Summary of pre-course material - Hui-Walter models

CA18208 HARMONY Serbia Training School - https://harmony-net.eu/

Eleftherios Meletis Julio Alvarez

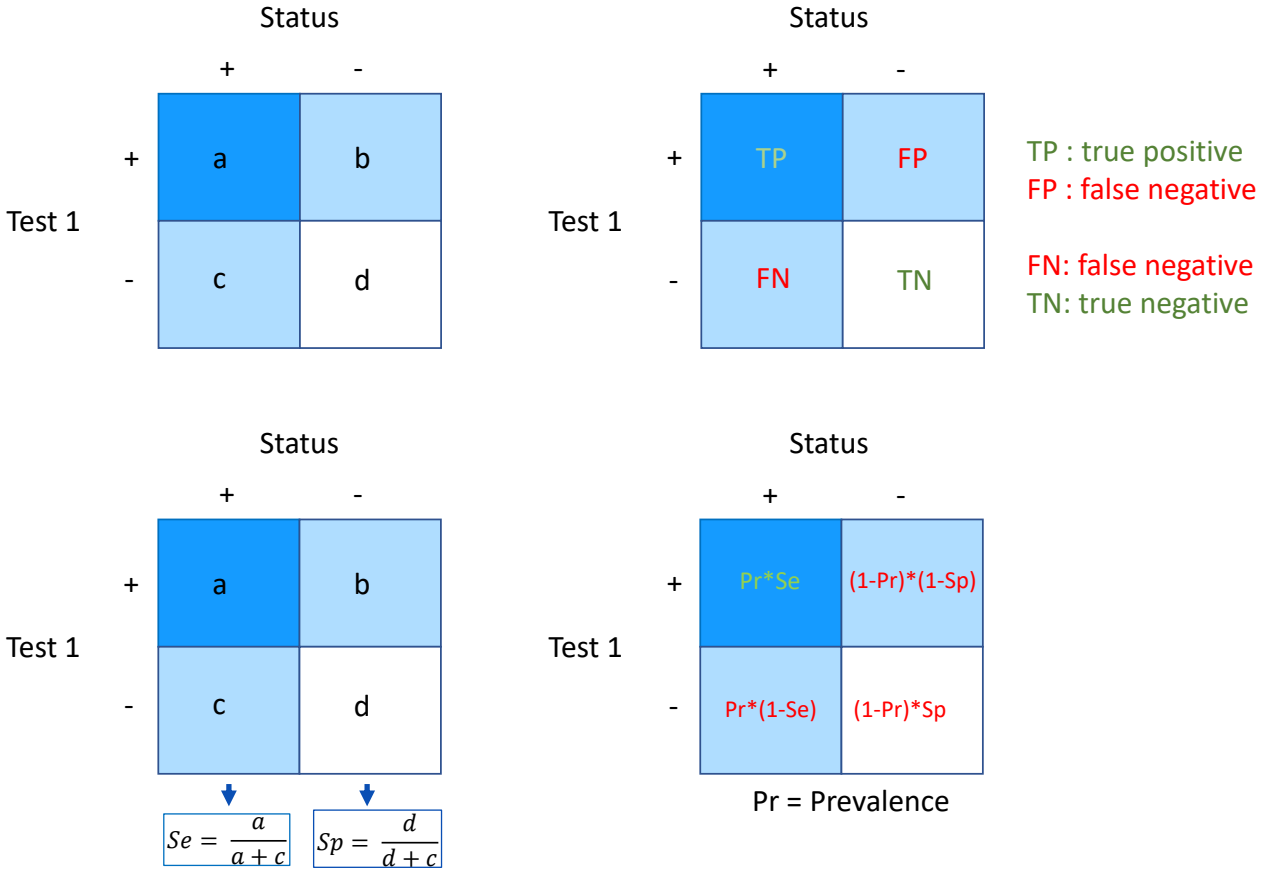
2022-08-31

Hui-Walter models

- A particular model formulation that was originally designed for evaluating diagnostic tests in the absence of a gold standard
- Not originally/necessarily Bayesian - implemented using Maximum Likelihood
- But evaluating an imperfect test against another imperfect test is a bit like pulling a rabbit out of a hat
  - If we don't know the true disease status, how can we estimate sensitivity or specificity for either test?

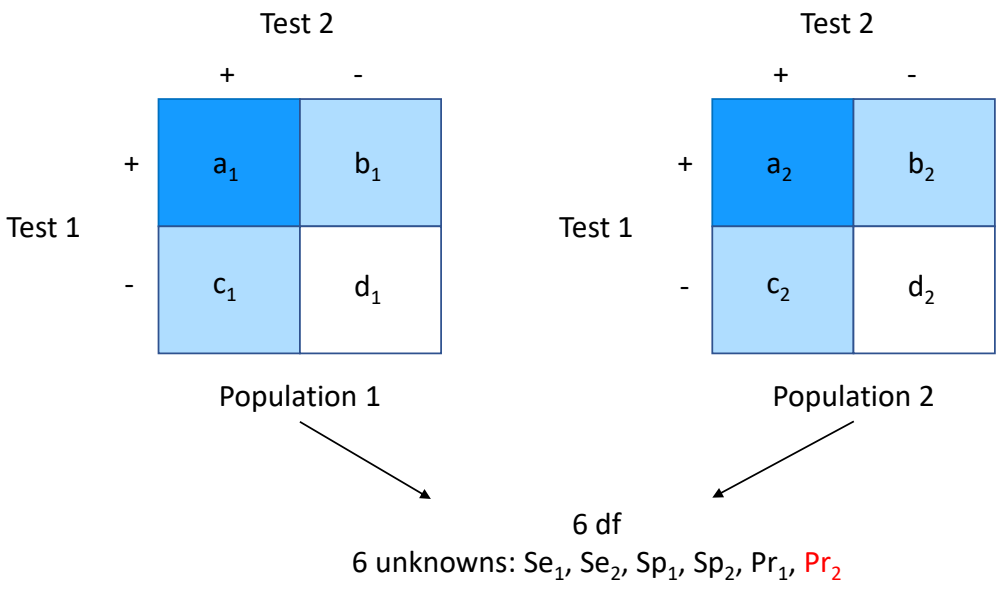
Se - *Sp* estimation - Recap

When the true infectious status is known *Se*-*Sp* of test 1 can be estimated:



Hui-Walter models (I)

- A particular model formulation that was originally designed for evaluating diagnostic tests in the absence of a gold standard
- Also known as the two_test - two_population setting/paradigm



Hui-Walter models (II)

In the Hui-Walter paradigm a condition that *connects* S the number of populations (P) and R the number of tests (T) is desribed:

$$S \geq \frac{R}{(2^{\text{R-1}} - 1)}$$

If this condition is fulfilled, any combination of S and R may allow to estimate Se and Sp, e.g. (2T, 2P), (3T,1P), (4T,1P), ...

- This condition describes **Model Identifiability** and is a necessary but not sufficient condition - as we'll see later.

Hui-Walter models

- For demonstration purposes we will start with the two_test - one_population setting (`hw_definition`) and continue adding both populations (S) and tests (R) along the way.

Two_test - One_population setting

- The data are summarized in a two_x_two table (2^2 cells)
- or in a vector that contains all possible test results combinations

Model Specification ('hw_definition')

```
model{
  Cross_Classified_Data ~ dmulti(prob, N)

  # Test1+ Test2+
    prob[1] <- (prev * ((se[1])*(se[2]))) + ((1-prev) * ((1-sp[1])*(1-sp[2])))

  # Test1+ Test2-
    prob[2] <- (prev * ((se[1])*(1-se[2]))) + ((1-prev) * ((1-sp[1])*(sp[2])))

  # Test1- Test2+
    prob[3] <- (prev * ((1-se[1])*(se[2]))) + ((1-prev) * ((sp[1])*(1-sp[2])))

  # Test1- Test2-
    prob[4] <- (prev * ((1-se[1])*(1-se[2]))) + ((1-prev) * ((sp[1])*(sp[2])))

  prev ~ dbeta(1, 1)
  se[1] ~ dbeta(1, 1)
  sp[1] ~ dbeta(1, 1)
  se[2] ~ dbeta(1, 1)
  sp[2] ~ dbeta(1, 1)

  #data# Cross_Classified_Data, N
  #monitor# prev, prob, se, sp
  #inits# prev, se, sp
}
```

```r
twoXtwo <- matrix(c(36, 4, 12, 48), ncol=2, nrow=2)
twoXtwo
##      [,1] [,2]
## [1,]   36   12
## [2,]    4   48
```

```r
library('runjags')

Cross_Classified_Data <- as.numeric(twoXtwo)
N <- sum(Cross_Classified_Data)

prev <- list(chain1=0.05, chain2=0.95)
se <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))
sp <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))

results <- run.jags('basic_hw.txt', n.chains=2)
```

Remember to check convergence and effective sample size!

Recap - psrf: potential scale reduction factor

- The psrf or Gelman-Rubin statistic is an estimated factor by which the scale of the current distribution for the target distribution might be reduced if the simulations were continued for an infinite number of iterations.
- psrf estimates the potential decrease in the between-chains variability with respect to the within-chain variability.
- If psrf is large, then longer simulation sequences are expected to either decrease between-chains variability or increase within-chain variability because the simulations have not yet explored the full posterior distribution.
- If psrf < 1.1 for all model parameters, one can be fairly confident that convergence has been reached. Otherwise, longer chains or other means for improving the convergence may be needed.

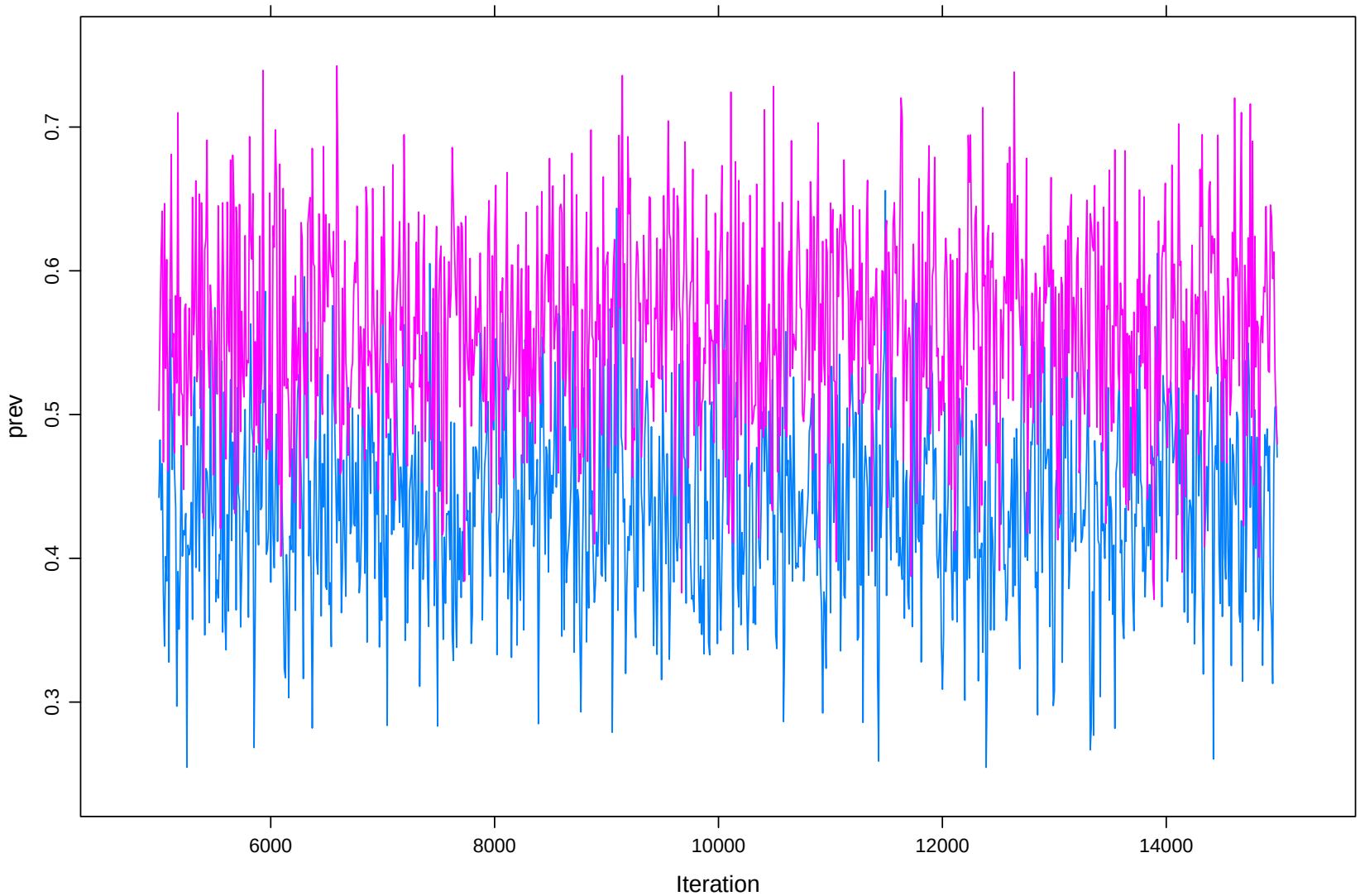
Recap - SSeff: effective sample size

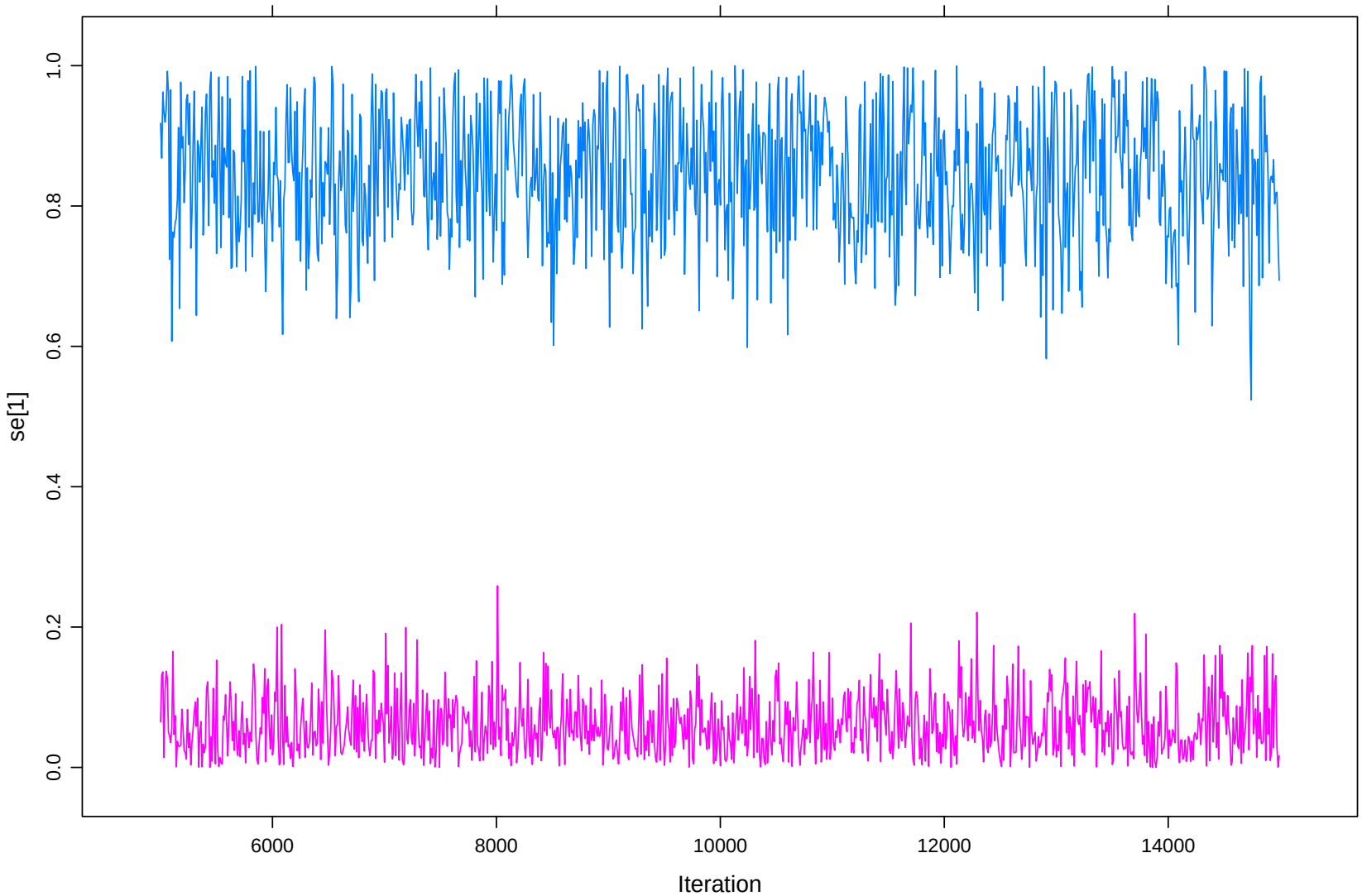
- Effective sample size is a calculation of the equivalent number of independent samples that would contain the same posterior accuracy as the correlated samples from an MCMC. Thus, MCMC Efficiency is the number of effectively independent samples generated per second.
- With high autocorrelation, the effective sample size will be lower.
- We want SSeff > 1000

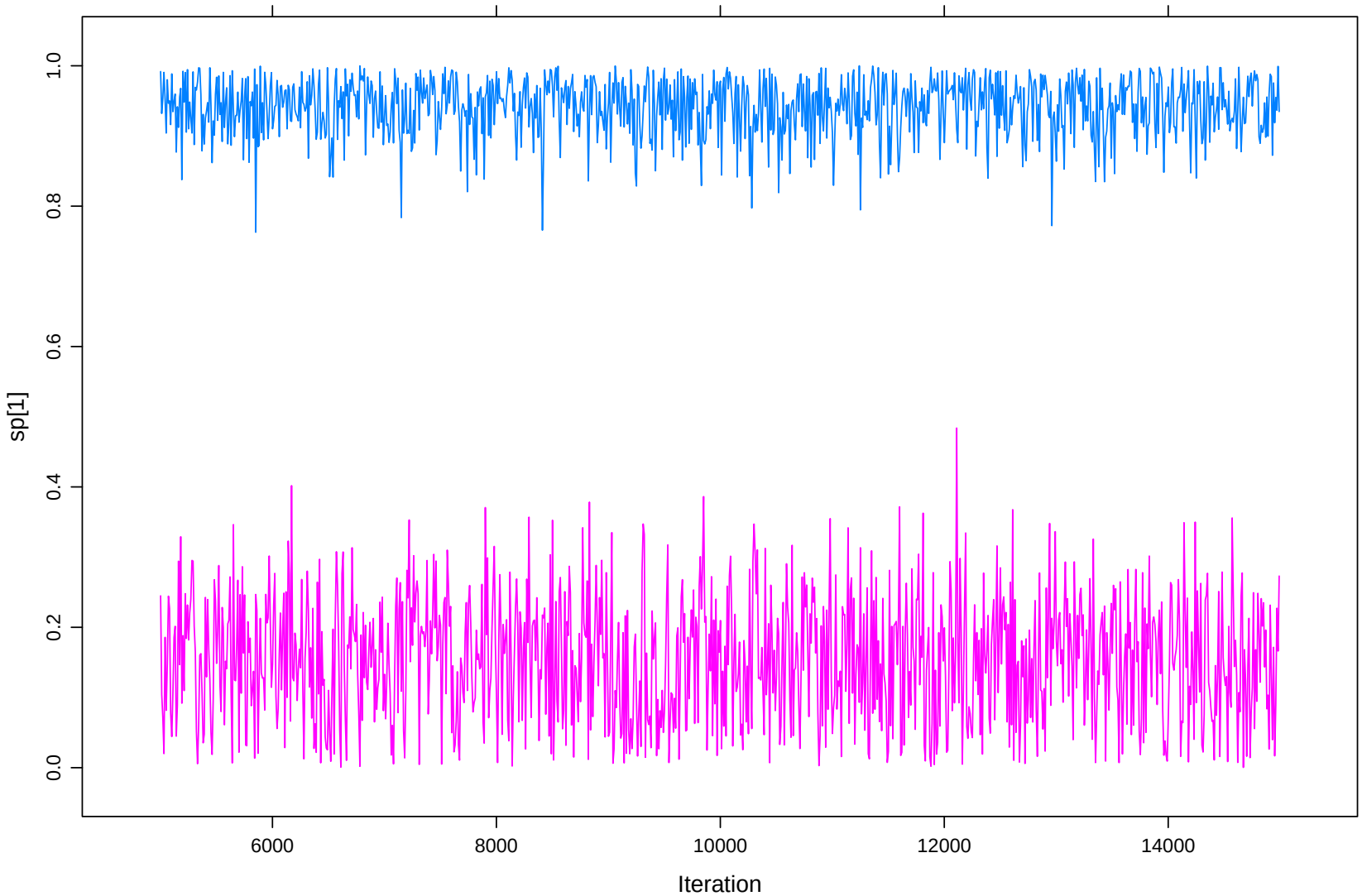
```
summary(results)
```

	Lower95	Median	Upper95	SSeff	psrf
prev	0.331	0.501	0.672	4300	2.308
prob[1]	0.254	0.344	0.435	14214	1.000
prob[2]	0.017	0.055	0.102	9934	1.000
prob[3]	0.073	0.133	0.200	14322	1.000
prob[4]	0.367	0.462	0.560	13327	1.000
se[1]	0.000	0.372	0.969	4470	13.603
se[2]	0.028	0.548	1.000	4622	15.105
sp[1]	0.034	0.618	1.000	4371	13.611
sp[2]	0.000	0.470	0.971	4672	14.975

```
plot(results)
```


prev
0.3
0.4
0.5
0.6
0.7
6000
8000
10000
12000
14000
Iteration


se[1]
1.0
0.8
0.6
0.4
0.2
0.0
6000
8000
10000
12000
14000
Iteration


sp[1]
0.0
0.2
0.4
0.6
0.8
1.0
6000
8000
10000
12000
14000
Iteration

- Does anybody spot a problem?

Label Switching

How to interpret a test with Se=0% and Sp=0%?

- The test is perfect - we are just holding it upside down...

We can force Se+Sp >= 1.

Jouden's Index Se + Sp -1 >= 0:

```
se[1] ~ dbeta(1, 1)
sp[1] ~ dbeta(1, 1)T(1-se[1], )
```

Or:

```
se[1] ~ dbeta(1, 1)T(1-sp[1], )
sp[1] ~ dbeta(1, 1)
```

This allows the test to be useless, but not worse than useless.

Alternatively we can have the weakly informative priors:

```
se[1] ~ dbeta(2, 1)
sp[1] ~ dbeta(2, 1)
```

To give the model some information that we expect the test characteristics to be closer to 100% than 0%.

Alternatively we can have the weakly informative priors:

```
se[1] ~ dbeta(2, 1)
sp[1] ~ dbeta(2, 1)
```

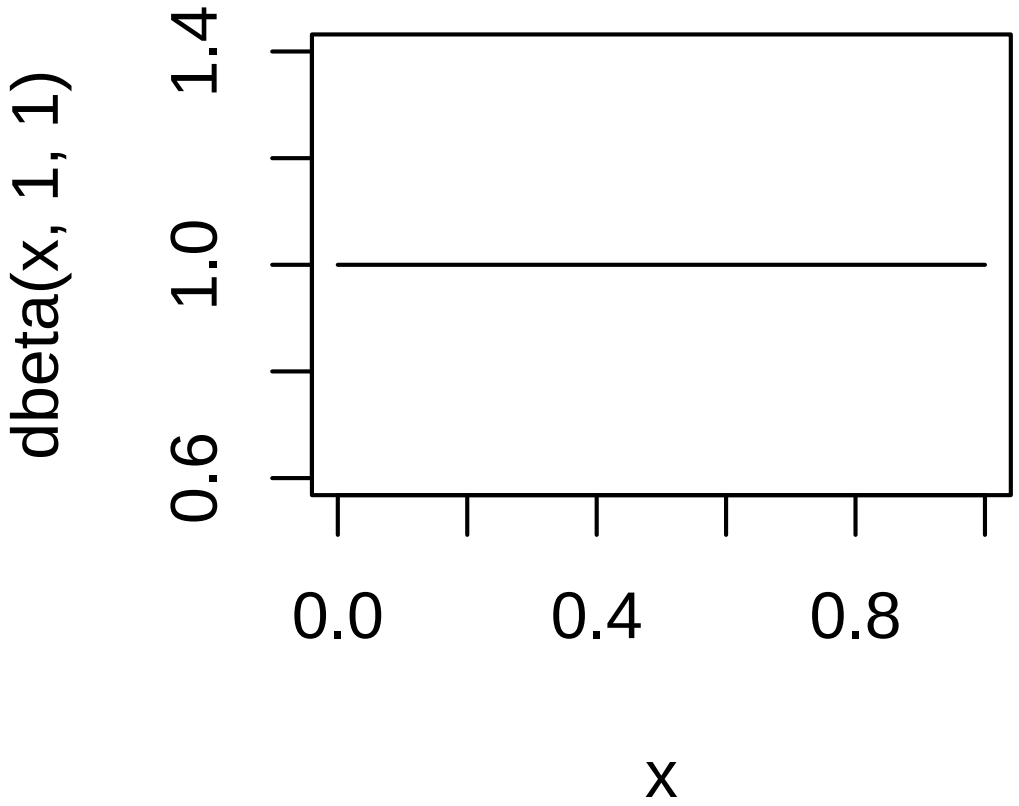
To give the model some information that we expect the test characteristics to be closer to 100% than 0%.

Or we can use stronger priors for one or both tests.

Priors

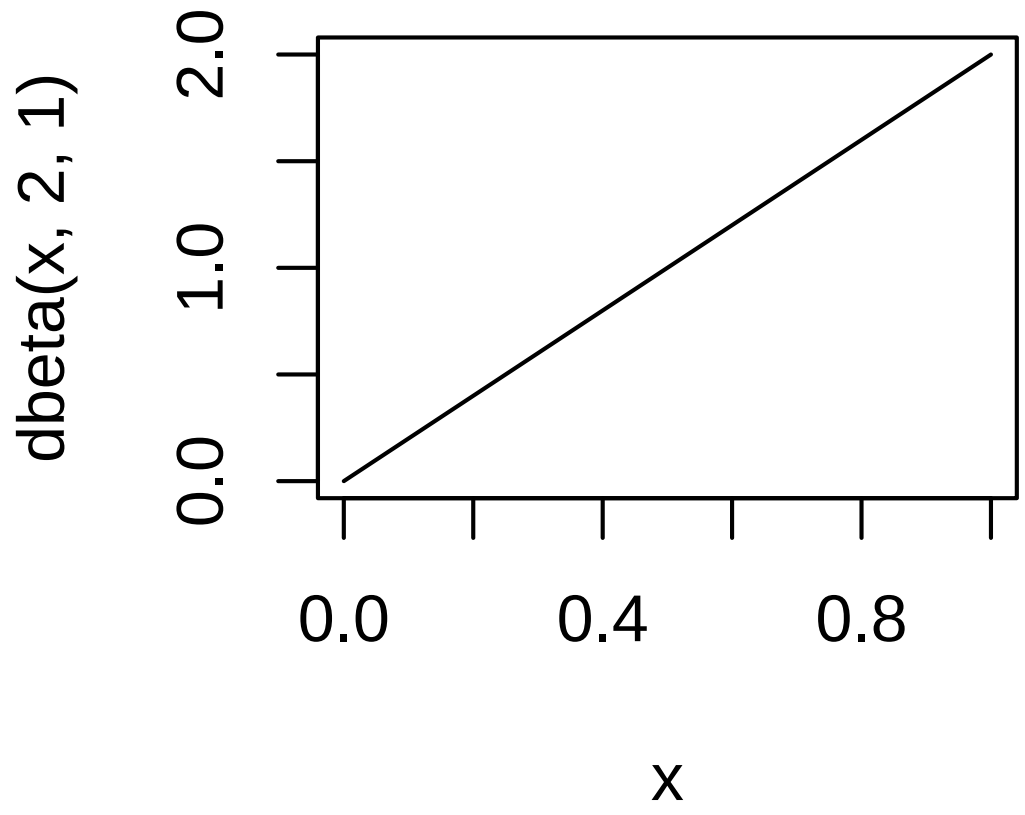
A quick way to see the distribution of a prior:

```
curve(dbeta(x, 1, 1), from=0, to=1)
```



This was minimally informative, but how does that compare to a weakly informative prior for e.g. sensitivity?

```
curve(dbeta(x, 2, 1), from=0, to=1)
```



Choosing a prior

What we want is e.g. Beta(20,1)

But typically we have median and 95% confidence intervals from a paper, e.g.:

"The median (95% CI) estimates of the sensitivity and specificity of the shiny new test were 94% (92-96%) and 99% (97-100%) respectively"

- How can we generate a Beta(,) prior from this?

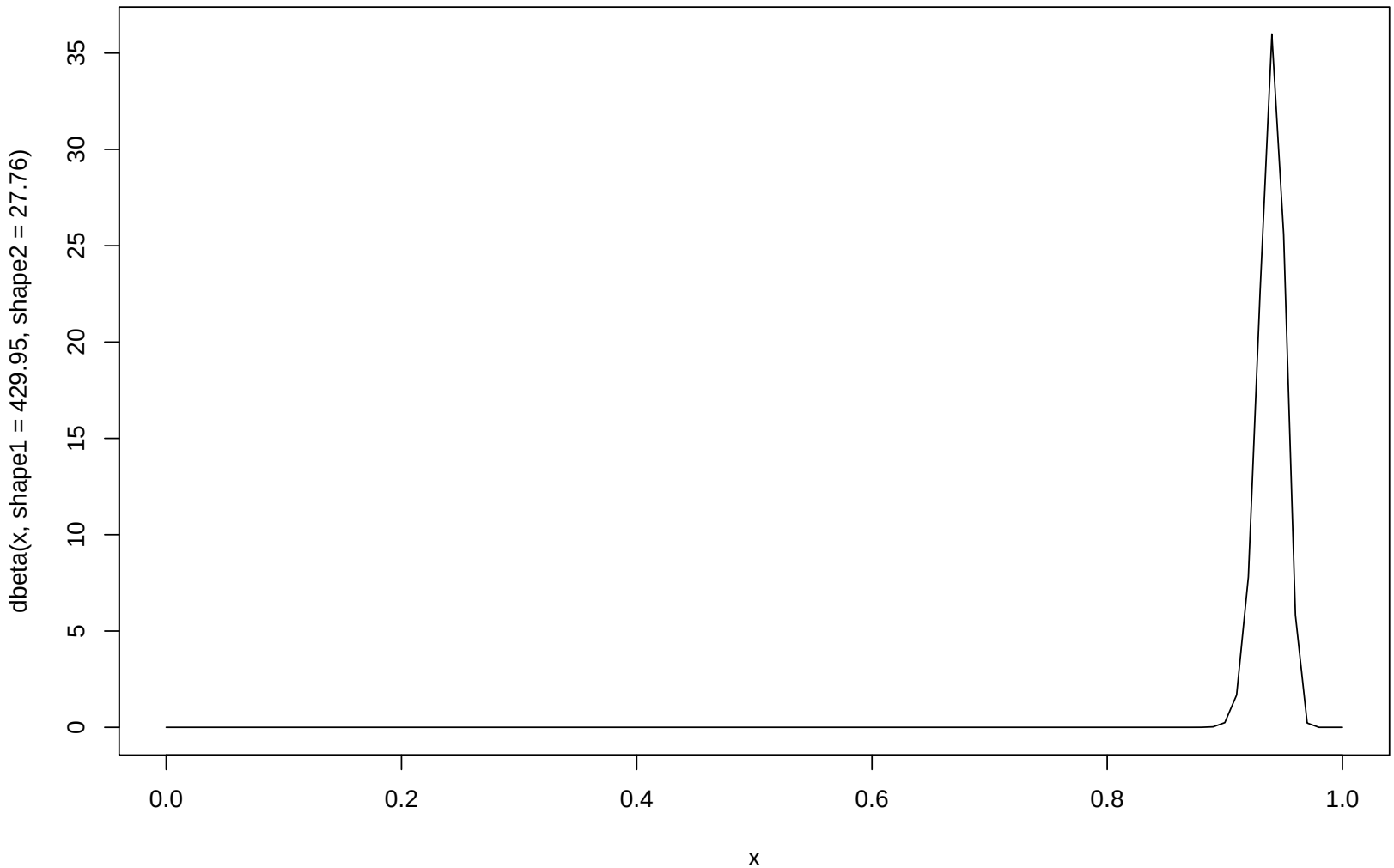
The PriorGen package

Median (95% CI) estimates of Se and Sp were 94% (92-96%) and 99% (97-100%)

```
library("PriorGen")
## Loading required package: rootSolve
findbeta(themedian = 0.94, percentile=0.95, percentile.value = 0.92)
## [1] "The desired Beta distribution that satisfies the specified conditions is
## [1] "Here is a plot of the specified distribution."
## [1] "Descriptive statistics for this distribution are:"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8807  0.9322  0.9398  0.9392  0.9472  0.9753
## [1] "Verification: The percentile value 0.92 corresponds to the 0.05 th perce
```

Note: `themedian` could also be `themean`

```
curve(dbeta(x, shape1=429.95, shape2=27.76))
```



Initial values

Part of the problem before was also that we were specifying extreme initial values:

```
se <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))
sp <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))
```

Initial values

Part of the problem before was also that we were specifying extreme initial values:

```
se <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))
sp <- list(chain1=c(0.01,0.99), chain2=c(0.99,0.01))
```

Let's change these to:

```
se <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
sp <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
```

Exercise 1

Run the `hw_definition` model under the following different scenarios and interpret the results in each case.

1. Change the priors for *Se* and *Sp* and try Beta(2,1).
2. Estimate the Beta parameters for the *Se* and *Sp* of the shiny new test that is described in the slides.
3. Run the model using the beta priors for *Se* and *Sp* from Step 2.
4. Try to run the model with different initial values.
5. Force Se + Sp $>$ 1. Be careful to specify initial values that are within the restricted parameter space.

Time for our lunch break

Multi-population Hui-Walter models

Hui-Walter models with multiple populations

- Basically an extension of the single-population model
- Works best with multiple populations each with differing prevalence
  - Including an unexposed population works well
  - BUT be wary of assumptions regarding constant sensitivity/specificity across populations with very different types of infections

Different prevalence in different populations (1st assumption)

What changes?

- In each population the data are summarized in a two_x_two table (2^2 cells) again
- or each population has a vector that contains all possible test results combinations

Model specification

```
model{
  for(p in 1:Populations){
    Tally[1:4, p] ~ dmulti(prob[1:4, p], TotalTests[p])
    # Test1- Test2- Pop1
    prob[1, p] <- (prev[p] * ((1-se[1])*(1-se[2])) + ((1-prev[p]) * (sp[1]*sp[2
    ## snip ##

    prev[p] ~ dbeta(1, 1)
  }

  se[1] ~ dbeta(se_prior[1,1], se_prior[1,2])T(1-sp[1], )
  sp[1] ~ dbeta(sp_prior[1,1], sp_prior[1,2])
  se[2] ~ dbeta(se_prior[2,1], se_prior[2,2])T(1-sp[2], )
  sp[2] ~ dbeta(sp_prior[2,1], sp_prior[2,2])

  #data# Tally, TotalTests, Populations, se_prior, sp_prior
  #monitor# prev, prob, se, sp
  #inits# prev, se, sp
}
```

Multiple populations: 2nd assumption

- We typically assume that the sensitivity and specificity *must* be consistent between populations
  - Do you have an endemic and epidemic population?
  - Or vaccinated and unvaccinated?
  - If so then the assumptions might not hold!

Multiple populations: 2nd assumption

- We typically assume that the sensitivity and specificity *must* be consistent between populations
  - Do you have an endemic and epidemic population?
  - Or vaccinated and unvaccinated?
  - If so then the assumptions might not hold!
- The populations can be artificial (e.g. age groups) but must not be decided based on the diagnostic test results
  - It helps if the prevalence differs between the populations

Multiple populations: special cases

- A small disease-free group is extremely helpful
  - Contains strong data regarding specificity
  - As long as specificity can be assumed to be the same in the other populations

Multiple populations: special cases

- A small disease-free group is extremely helpful
  - Contains strong data regarding specificity
  - As long as specificity can be assumed to be the same in the other populations
- A small experimentally infected group MAY be helpful but it is often dangerous to assume that sensitivity is consistent!
  - *Se* could differ based (I) age (II) epidemiological contex (III) stage of infection...

Initial values

We have to be careful to make sure that the length of initial values for `prev` in each chain is equal to the number of populations

For example with 5 populations we need:

```
prev <- list(chain1=c(0.1, 0.1, 0.1, 0.9, 0.9), chain2=c(0.9, 0.9, 0.9, 0.1, 0.
```

The values you choose for different populations in the same chain can be the same - just make sure you pick different values for the same population between chains (i.e. *over-dispersed* initial values)

Note: specifying your own initial values is technically optional with JAGS, but it is always a good idea (for now at least)!!!

Incorporating populations with known prevalence

Up to now prevalence has been a parameter, but it can also be (partially) observed:

```
model{
  for(p in 1:Populations){
    Tally[1:4, p] ~ dmulti(prob[1:4, p], TotalTests[p])
    # Test1- Test2- Pop1
    prob[1, p] <- (prev[p] * ((1-se[1])*(1-se[2])) + ((1-prev[p]) * (sp[1]*sp[2
    ## snip ##

    prev[p] ~ dbeta(1, 1)
  }

  ## snip ##

  #data# Tally, TotalTests, Populations, se_prior, sp_prior, prev
  #monitor# prev, prob, se, sp
  #inits# prev, se, sp
}
```

To fix the prevalence of population 1 we could do:

```
Populations <- 5
prev <- rep(NA, Populations)
prev[1] <- 0
prev
## [1]  0 NA NA NA NA
```

To fix the prevalence of population 1 we could do:

```
Populations <- 5
prev <- rep(NA, Populations)
prev[1] <- 0
prev
## [1]  0 NA NA NA NA
```

But you also need to account for this in the initial values:

```
prev <- list(chain1=c(NA, 0.1, 0.1, 0.9, 0.9), chain2=c(NA, 0.9, 0.9, 0.1, 0.1)
```

Note: we now have two definitions of `prev` in R!

Data and initial value lists

There are actually multiple ways to specify data and initial values to runjags, including via the `data` and `inits` arguments

We will use these to keep separate lists of data and initial values (these could also be data frames, or environments)

```
data <- list(
  Tally = Tally,
  TotalTests = apply(Tally, 2, sum),
  Populations = dim(Tally, 2),
  prev = rep(NA, Populations),
  se_prior = matrix(1, ncol=2, nrow=2),
  sp_prior = matrix(1, ncol=2, nrow=2)
)
data$prev[1] <- 0
```

```
inits <- list(
  chain1 = list(
    prev = c(NA, 0.1, 0.1, 0.9, 0.9),
    se = c(0.5, 0.99),
    sp = c(0.5, 0.99)
  ),
  chain2 = list(
    prev = c(NA, 0.9, 0.9, 0.1, 0.1),
    se = c(0.99, 0.5),
    sp = c(0.99, 0.5)
  )
)

results <- run.jags(..., data = data, inits = inits)
```

See the help file for `?run.jags` for more details

Let's simulate a dataset to work on. . .

```r
# Set a random seed so that the data are reproducible:
set.seed(2022-08-31)

sensitivity <- c(0.9, 0.6)
specificity <- c(0.95, 0.9)
N <- 1000

# Change the number of populations here:
Populations <- 5
# Change the variation in prevalence here:
(prevalence <- runif(Populations, min=0.1, max=0.9))

data <- tibble(Population = sample(seq_len(Populations), N, replace=TRUE)) %>%
  mutate(Status = rbinom(N, 1, prevalence[Population])) %>%
  mutate(Test1 = rbinom(N, 1, sensitivity[1]*Status + (1-specificity[1])*(1-Sta
  mutate(Test2 = rbinom(N, 1, sensitivity[2]*Status + (1-specificity[2])*(1-Sta

(twoXtwoXpop <- with(data, table(Test1, Test2, Population)))
(Tally <- matrix(twoXtwoXpop, ncol=Populations))
(TotalTests <- apply(Tally, 2, sum))
```

```
Tally
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   77  123  120   60   41
## [2,]   43   23   30   55   61
## [3,]   11   20   18   13   12
## [4,]   51   20   39   73  110
```

Exercise 2

Start with 5 populations and analyse the data using the independent prevalence model.

Now try with 2, 3, and 10 populations.

- How does this affect the confidence intervals for the diagnostic test parameters?

Now change the simulated prevalence so that it varies between 0.4-0.6 rather than 0.1-0.9.

- How does this affect the confidence intervals for the diagnostic test parameters?

Solution 2

This is what the model should look like:

```
model{
  for(p in 1:Populations){
    Tally[1:4, p] ~ dmulti(prob[1:4, p], TotalTests[p])

    # Test1- Test2-
          prob[1,p] <- (prev[p] * ((1-se[1])*(1-se[2]))) + ((1-prev[p]) * ((sp[
    # Test1+ Test2-
    prob[2,p] <- (prev[p] * ((se[1])*(1-se[2]))) + ((1-prev[p]) * ((1-sp[1])*(s
    # Test1- Test2+
    prob[3,p] <- (prev[p] * ((1-se[1])*(se[2]))) + ((1-prev[p]) * ((sp[1])*(1-s
     # Test1+ Test2+
     prob[4,p] <- (prev[p] * ((se[1])*(se[2]))) + ((1-prev[p]) * ((1-sp[1])*(1-

    prev[p] ~ dbeta(1, 1)
  }
  se[1] ~ dbeta(se_prior[1,1], se_prior[1,2])T(1-sp[1], )
  sp[1] ~ dbeta(sp_prior[1,1], sp_prior[1,2])
  se[2] ~ dbeta(se_prior[2,1], se_prior[2,2])T(1-sp[2], )
  sp[2] ~ dbeta(sp_prior[2,1], sp_prior[2,2])

  #data# Tally, TotalTests, Populations, se_prior, sp_prior
  #monitor# prev, se, sp
  #inits# prev, se, sp
  #module# lecuyer
```

Here is the R code to run the model:

```
set.seed(2022-08-31)

# Set up the se_prior and sp_prior variables (optional):
se_prior <- matrix(1, nrow=2, ncol=2)
sp_prior <- matrix(1, nrow=2, ncol=2)

# Set up initial values for 5 populations:
se <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
sp <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
prev <- list(chain1=c(0.1, 0.1, 0.1, 0.9, 0.9), chain2=c(0.9, 0.9, 0.9, 0.1, 0.

# And run the model:
results_5p <- run.jags("multipopulation.txt", n.chains=2)

# Remember to check convergence!
# plot(results_5p)
```

```
summary(results_5p)
##             Lower95    Median   Upper95      Mean          SD Mode         MCerr
## prev[1] 0.38877508 0.4919613 0.5904441 0.4910855 0.05162704   NA 0.0008294940
## prev[2] 0.08788326 0.1724590 0.2600808 0.1744986 0.04458009   NA 0.0008015901
## prev[3] 0.20496228 0.2931453 0.3806864 0.2935081 0.04562961   NA 0.0007833312
## prev[4] 0.54177629 0.6380353 0.7323108 0.6372005 0.04833300   NA 0.0008062360
## prev[5] 0.70944214 0.7899621 0.8660228 0.7890607 0.04016314   NA 0.0006730447
## se[1]   0.88475495 0.9406139 0.9958233 0.9392102 0.02948724   NA 0.0005212590
## se[2]   0.56588179 0.6249711 0.6888099 0.6255467 0.03128200   NA 0.0005517510
## sp[1]   0.83723882 0.9083274 0.9842500 0.9084172 0.03753432   NA 0.0007970405
## sp[2]   0.83709666 0.8772701 0.9194724 0.8768853 0.02113101   NA 0.0003269900
##         MC%ofSD SSeff      AC.10      psrf
## prev[1]     1.6  3874 0.04611107 0.9999982
## prev[2]     1.8  3093 0.08131069 1.0002205
## prev[3]     1.7  3393 0.08179315 1.0004230
## prev[4]     1.7  3594 0.06161511 1.0007013
## prev[5]     1.7  3561 0.07156928 1.0003239
## se[1]       1.8  3200 0.04960371 1.0010077
## se[2]       1.8  3214 0.06797831 0.9999581
## sp[1]       2.1  2218 0.11828309 1.0000652
## sp[2]       1.5  4176 0.04568133 1.0012669
```

To change the number of populations and range of prevalence you just need to modify the simulation code, for example 3 populations with prevalence between 0.4-0.6 can be obtained using:

```
# Change the number of populations here:
Populations <- 3
# Change the variation in prevalence here:
(prevalence <- runif(Populations, min=0.4, max=0.6))

data <- tibble(Population = sample(seq_len(Populations), N, replace=TRUE)) %>%
  mutate(Status = rbinom(N, 1, prevalence[Population])) %>%
  mutate(Test1 = rbinom(N, 1, sensitivity[1]*Status + (1-specificity[1])*(1-Sta
  mutate(Test2 = rbinom(N, 1, sensitivity[2]*Status + (1-specificity[2])*(1-Sta

(twoXtwoXpop <- with(data, table(Test1, Test2, Population)))
(Tally <- matrix(twoXtwoXpop, ncol=Populations))
(TotalTests <- apply(Tally, 2, sum))
# Adjust initial values for 3 populations:
se <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
sp <- list(chain1=c(0.5,0.99), chain2=c(0.99,0.5))
prev <- list(chain1=c(0.1, 0.1, 0.9), chain2=c(0.9, 0.9, 0.1))
# And run the model:
results_3p <- run.jags("multipopulation.txt", n.chains=2)
# Remember to check convergence!
# plot(results_3p)
# summary(results_3p)
```

Note that when the effective sample size is not enough - you either need to run the model for longer in the first place, or extend it to get more samples:

```
# Extend the model:
results_3p <- extend.jags(results_3p, sample=50000)

# Remember to check convergence!
# plot(results_3p)
# summary(results_3p)
```

As a general rule, the more populations you have, and the more the prevalence varies between them, the better. However, this is conditional on having a consistent sensitivity and specificity between your populations!!!

Day 1 Summary