MCMC with Stan

Alberto Garfagnini

Università di Padova

AA 2023/2024 - Stat Lect 11

Stan

- Stan is another program that allows to specify statistical models, and to sample from a posterior distribution
- it is named after Stanislaw Ulam (1909-1984), a pioneer of Monte Carlo methods
- Stan means also Sampling Through Adaptive Neighborhoods
- it uses a different method than JAGS for generating Monte Carlo steps, called Hamiltonian Monte Carlo (HMC)
- HMC uses a sampling scheme creating proposal distributions pulled toward the modes of the posterior distribution instead of being symmetrical around the current position

References

- Stan web site: `https://mc-stan.org/`
- R interfaces to Stan:

  rstan: the R interface to Stan, `https://github.com/stan-dev/rstan`

  brms: an interface to fit Bayesian generalized non-linear multivariate multilevel models using Stan, `https://github.com/paul-buerkner/brms`

Using Stan

- to use Stan, the user writes a Stan program representing the statistical model
- the user specifies:
  - the parameters in the model
  - the target posterior density
- Stan code is compiled, run along with the data, and it provides a set of posterior simulations of the parameters
- Stan is written in C++ and interfaces with several popular data analysis languages are available (R, Python, MATLAB, Julia, . . .)

Stan program structure

- a stan program is organized in blocks
- all blocks are optional ➜ an empty string is a valid Stan program, even if it will trigger a warning message from the Stan compiler
- detailed explanations on the Stan language can be found here: `https://mc-stan.org/docs/reference-manual/`
- the three basics blocks are the following:

  for the declaration of variables that are read in as data

```
data {
  //  ... data ...
}
```

  these are the variables being sampled by Stan's samplers

```
parameters {
   // ... declarations ...
}
```

  it define the model. Here probability statements are given

```
model {
   // ... declarations ... statements ...
}
```

Stan program structure

- a stan program is organized in blocks

```
functions {
  // ... function declarations and definitions ...
}


transformed data {
   // ... declarations ... statements ...
}


transformed parameters {
   // ... declarations ... statements ...
}
```

Chains initialization

- JAGS and Stan can automatically start the MCMC chains at default values
- but the efficiency of the MCMC process can be improved using appropriate starting values
- guideline: figure out values for the parameters in the model that are a reasonable description of the data, and of the posterior distribution
- a good choice: maximum likelihood estimate (MLE) of the parameters : i.e. maximize the probability of the data
- another approach is to start the chains at random points near the MLE

Example: Bernoulli trial

- the MLE of the parameter is $p = y/N$
- it maximizes $p^y(1-p)^{N-y}$

Three ways to initialize a chain in JAGS/Stan

- a single named list with a single initial point for the parameters ➜ all chains start there
- a list of lists, with as many sub-lists as chains ➜ with specific initial values in each sub-list
- define a function that returns initial values when called

rstan

Running stan in R

```
library(rstan)

#> Loading required package: StanHeaders
#> Loading required package: ggplot2
#> rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)
#> For execution on a local, multicore CPU with excess RAM we recommend call
#> options(mc.cores = parallel::detectCores()).
#> To avoid recompilation of unchanged Stan programs, we recommend calling
#> rstan_options(auto_write = TRUE)
```

- as the startup message says, if you are using rstan locally on a multicore machine and have plenty of RAM to estimate your model in parallel, at this point execute

```
options(mc.cores = parallel::detectCores())

# on my machine:
parallel::detectCores()
#> [1] 12
```

- to avoid recompiling of C++ code every time (unless there are changes)

```
rstan_options(auto_write = TRUE)
```

Bernoulli example: JAGS

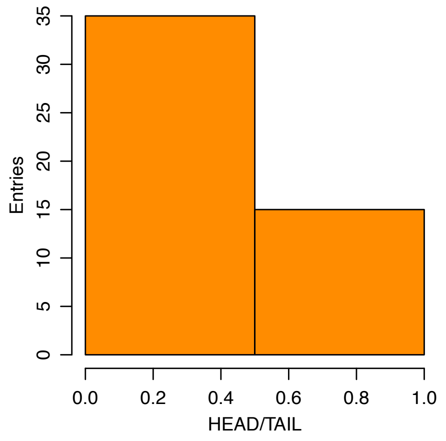
1) define the model and write it into a file

```
modelString = "
  model {
    for ( i in 1:Ntotal ) {
      y[i] ~ dbern( theta )
    }
    theta ~ dbeta( 1 , 1 )
  }
"
writeLines(modelString , con="jags_bern01_model.txt")
```

```
cat bern_jags-stan.csv
"y"
0
1
...
1
0
```

2) import data from a file and create a named-list

```
myData <- read.csv("bern_jags-stan.csv")
y <- myData$y
Ntotal <- length(y)
dataList <- list(y = y, Ntotal = Ntotal)
```



3) gets all the information into JAGS and lets it figure out appropriate samplers for the model

```
jagsModel <- jags.model(file="jags_model.txt",
                         data=dataList,
                         # inits=initsList,
                         n.chains=3,
                         n.adapt=500)
```

Bernoulli example: JAGS

4) run the chains for a burn-in period

```
update(jagsModel, n.iter=500)
```

the `update` function returns no values.
It does not record the sampled parameter values during the updating

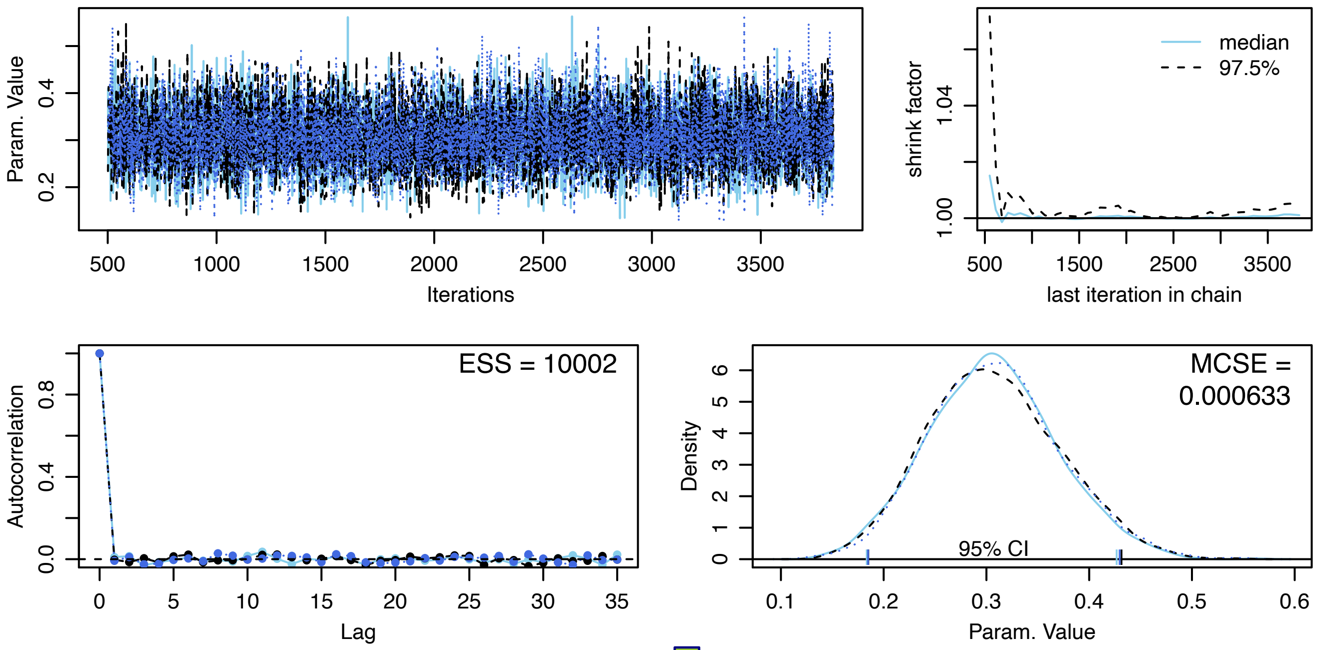
5) let JAGS generate MCMC samples that will be used to represent the posterior distribution

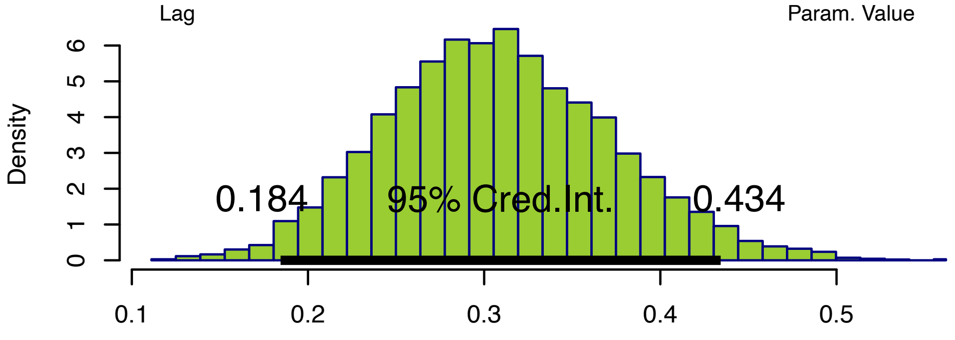
```
codaSamples <- coda.samples(jagsModel, variable.names=c("theta") ,
                            n.iter=3334)
```

the chains are arranged in a specialized format so that various functions from the coda package can be used to examine the chains

`variable.names` argument must be a vector of character strings

Bernoulli example: JAGS





Bernoulli example: JAGS

- the effective sample size is computed by coda by summing across chains

```
postSummary[,"ESS"] <- coda::effectiveSize(sampleVec)
```

- we can compute mean and median and the mode through the `density` function (which computes kernel density estimates)

```
postSummary[,"mean"]   <- mean(sampleVec)
postSummary[,"median"] <- median(sampleVec)

mcmcDensity <- density(sampleVec)
postSummary[,"mode"] <- mcmcDensity$x[which.max(mcmcDensity$y)]
```

- Credibility Interval:

```
sPts <- sort(sampleVec)
cIdx_step <- ceiling(credInt * length(sPts))
nCIs <- length(sPts) - ciIdx_step
ciWidth <- rep(0, nCIs)
for (j in 1:nCIs) {
  ciWidth[i] <- sPts[ j + ciIdx_step ] - sPts[j]
}
HDImin = sPts[ which.min( ciWidth ) ]
HDImax = sPts[ which.min( ciWidth ) + ciIdx_step ]
HDIlim = c( HDImin , HDImax )
```

Bernoulli example: Stan

1) define the model and write it into a file

   note that every line ends with ';' ➜ this is the C++ syntax, used in Stan

```
modelString = "
  data {
    int<lower=0> N;
    int y[N];
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  model {
    theta ~ beta(1,1);
    y ~ bernoulli(theta);
  }"
writeLines(modelString , con="stan_bern01_model.txt")
```

Note: Stan allows and encourages vectorization of operations.

a single line can indicate that every y_i follows the Bernoulli distribution:

```
y ~ bernoulli(theta);
```

Bernoulli example: Stan

2) translate the model to Stan C++ *Dynamic Shared Object (DSO) code*

```
stanDso <- stan_model(model_code = modelString)
```

once created, the DSO can be used for generating a Monte Carlo sample from the posterior distribution

3) *specify the data exactly as done for JAGS*

```
# Read the data and put it in a list
myData <- read.csv("bern_jags-stan.csv")
y <- myData$y    # The y values are in the column named y.
N <- length(y)   # Total number of coin flips
dataList <- list(y = y , N = N)
```

4) *generate the MC sample with the sampling command*

```
stanFit <- sampling(object=stanDso ,
                    data = dataList ,
                    chains = 3 ,
                    iter = 1000 ,
                    warmup = 200 ,
                    thin = 1)
```

Note: `warmup` *is used instead of* `burnin`

`iter` *is the total number of steps per chain*

Bernoulli example: Stan

3) RStan has methods for the standard R plot and summary commands and also its own version of the `traceplot` command

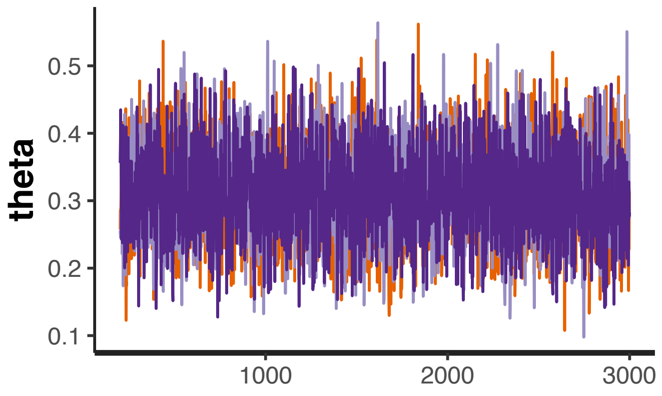
```
rstan::traceplot(stanFit,pars=c("theta"))
```

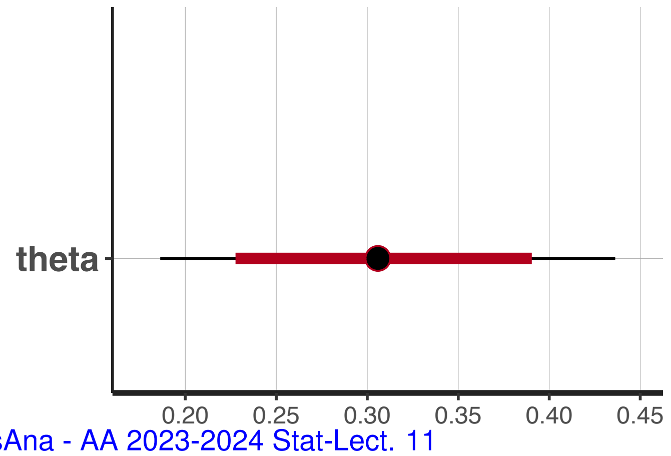
4) and a specialized plot version

```
plot(stanFit,pars=c("theta"))
```

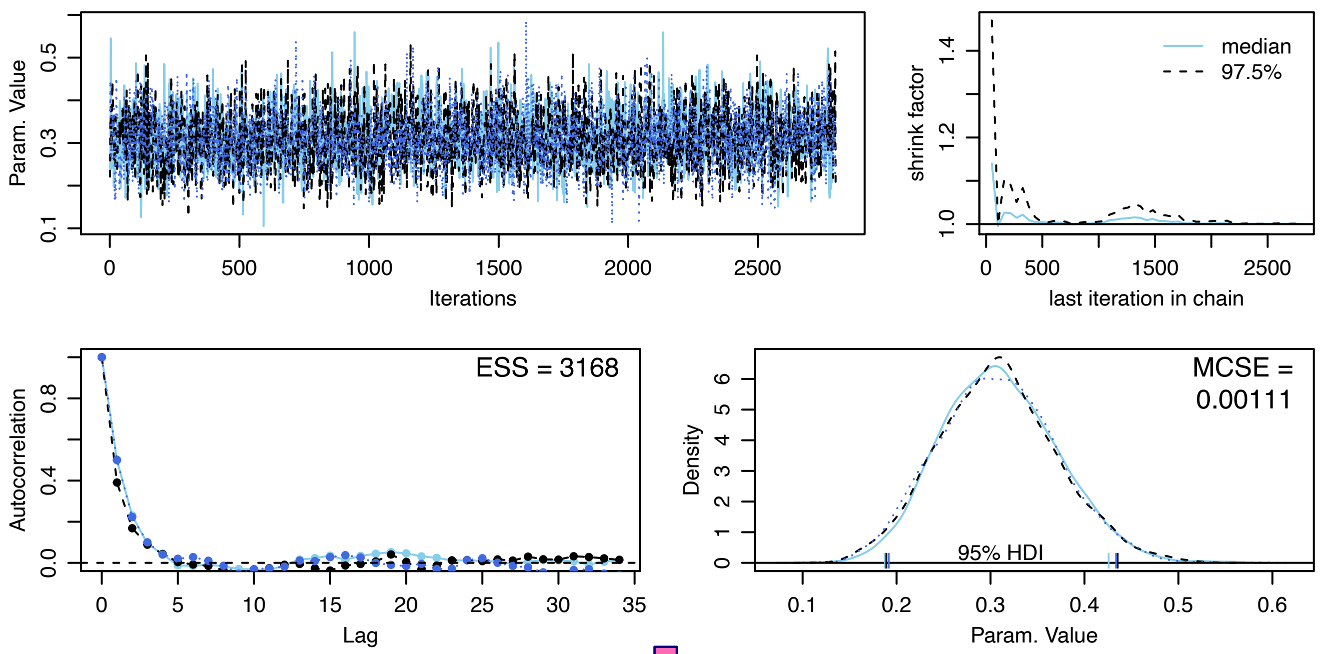
5) but it is always possible to transform the chain to a coda object list

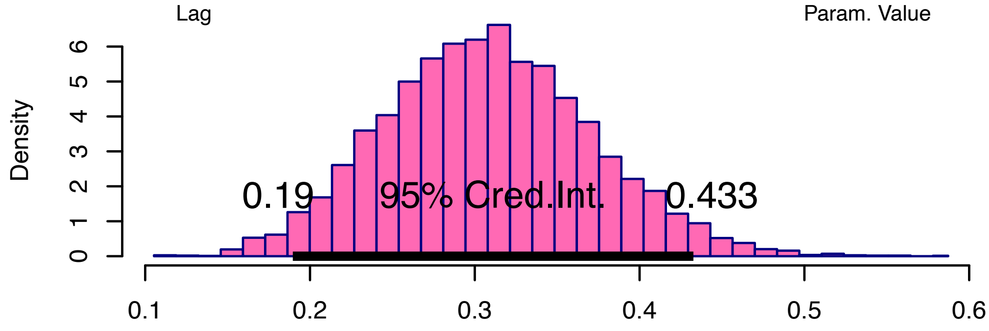
```
mcmcCoda <- mcmc.list(lapply(1:ncol(stanFit),
                      function(x) { mcmc(as.array(stanFit)[,x,]) }))
class(mcmcCoda)
#> [1] "mcmc.list"
```





Bernoulli example: Stan





Example: Vaccine effectiveness

Randomized controlled trials (RCTs)

- volunteers are assigned randomly to receive an influenza vaccine or a placebo
- vaccine efficacy is measured by comparing the frequency of influenza illness in the vaccinated and the unvaccinated (placebo) groups
- the RCT study design minimizes bias that could lead to invalid study results
- vaccine allocation is usually double-blinded : neither the volunteers nor the researchers know if a given person has received vaccine or placebo

Observational Studies

- compare the occurrence of influenza among people who have been vaccinated compared to people not vaccinated
- vaccine effectiveness is the percent reduction in the frequency of illness among vaccinated people compared to people not vaccinated
- adjustment for factors (like presence of chronic medical conditions) are considered

References

1) Center for Disease and Control Prevention:
`https://www.cdc.gov/flu/vaccines-work/effectivenessqa.htm`

2) European Center for Disease and Control Prevention:
`https://www.ecdc.europa.eu/en/covid-19/prevention-and-control/vaccines`

Pfizer example: `runjags`

1) Pfizer announced that their Vaccine against COVID-19 is more than 90% effective: https://www.npr.org/sections/health-shots/2020/11/09/933006651/pfizer-says-experimental-covid-19-vaccine-is-more-than-90-effective?t=1622093442237
   they studied 43538 volunteers and found 94 evaluable cases of COVID-19
   the American Food and Drug adimistration set a minimum effectiveness level at 50%: https://www.fda.gov/media/139638/download
2) collect and organize the data from RCT

```
tot_vaccine <- 21999
tot_placebo <- 21539
patient <- c(rep("Vaccine", tot_vaccine),
             rep("Placebo", tot_placebo))

# Number of patients tested postive after RCT:
pos_vaccine <- 8
pos_placebo <- 86
tested  <- c(rep("Pos", pos_vaccine),
             rep("Neg", tot_vaccine - pos_vaccine),
             rep("Pos", pos_placebo),
             rep("Neg", tot_placebo - pos_placebo))

pfizer.tb <- tibble(tested = tested, patient=patient)
table(pfizer.tb[[2]], pfizer.tb[[1]])
            Neg   Pos
  Placebo 21453    86
  Vaccine 21991     8
```

Pfizer example: `runjags`

3) define the JAGS model : we do not use a flat prior, since we have pretty good information on how likely is to get COVID. We use a `beta(3,100)` prior:

```
modelString <- "
  model {
    for ( i in 1:Ntot ) {
      tested[i] ~ dbern( theta[patient[i]] )
    }
    for ( k in 1:Nclass ) {
      theta[k] ~ dbeta(3 , 100)
    }
  }"
```

4) organize our data in a list for usage in JAGS

```
dataList = list(

   tested = ifelse(pfizer.tb$tested == "Neg", 0, 1),
   patient = as.integer(factor(pfizer.tb$patient)),

   Ntot = nrow(pfizer.tb) ,
   Nclass = nlevels(factor(pfizer.tb$patient))

)
```

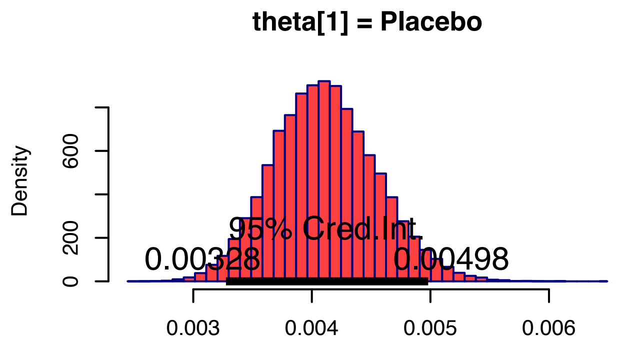
Pfizer example: `runjags`

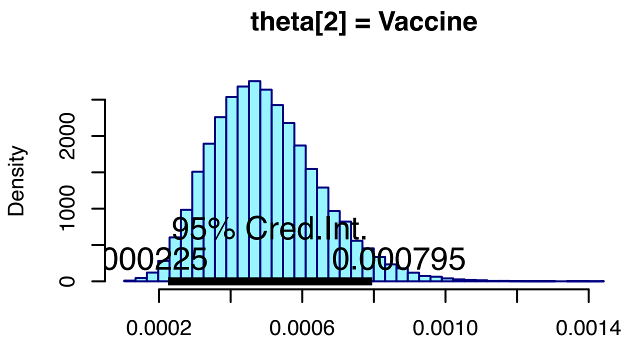
4) run JAGS

```
pfizer_chains <- run.jags(modelString,
                          sample = 15000,
                          n.chains = 4,
                          method = "parallel",
                          monitor = "theta",
                          data = dataList)
```

5) quick check JAGS run results:

```
summary(pfizer_chains)
             Lower95        Median      Upper95          Mean           SD  Mode
theta[1] 0.003294610 0.0041017350 0.004989380 0.0041159597 0.0004345924 NA
theta[2] 0.000223376 0.0004822135 0.000792961 0.0004975752 0.0001496030 NA
                MCerr  MC%ofSD SSeff          AC.10       psrf
theta[1] 2.172962e-06      0.5 40000 -0.002488248 0.9999861
theta[2] 7.389484e-07      0.5 40988 -0.004059210 1.0000766
```





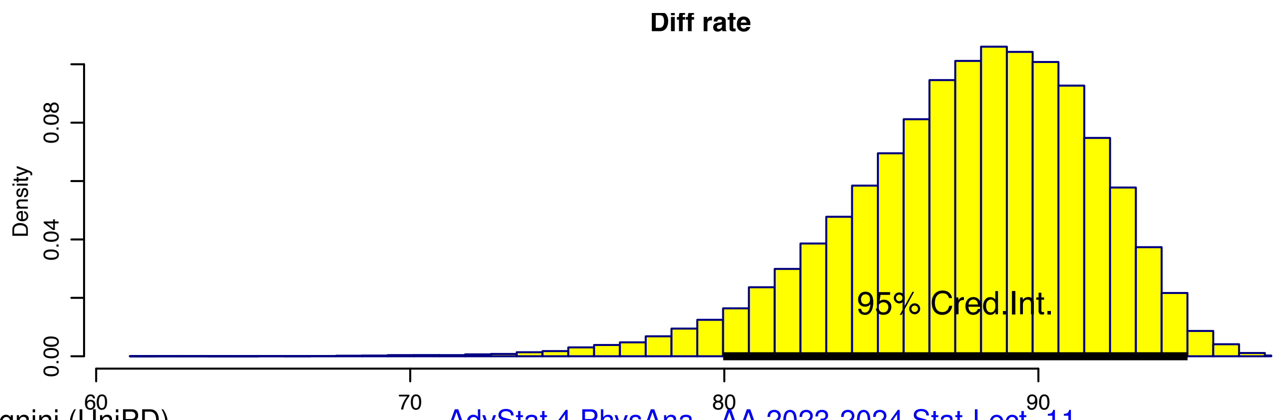
Pfizer example: `runjags`

1) have computed estimates for the rate of infection for both those who received the placebo and those who received the actual vaccine.
now want to investigate which is the percentage difference in infection rates

```
library(tidybayes)
pfizer_res   <- tidybayes::tidy_draws(pfizer_chains) %>%
                select(‘theta[1]‘:‘theta[2]‘) %>%
                rename(Placebo = ‘theta[1]‘, Vaccine = ‘theta[2]‘) %>%
                mutate(diff_rate = (Placebo - Vaccine) / Placebo * 100,
                Placebo_perc = Placebo * 100,
                Vaccine_perc = Vaccine * 100)
```

2) encapsulate the data in Coda so we can reuse our plotting function

```
allmcmc2 <- as.mcmc(pfizer_res, vars="diff_rate")
pt3 <- plotPosterior(allmcmc2[,"diff_rate"], 0.95, "yellow", "Diff␣rate")
             ESS     mean    median     mode CrIntLevel CrIntLow CrIntHigh
Param. Val. 60000 87.74049 88.17833 88.96141       0.95 79.96449  94.74905
```



Pfizer example: testing Bayes factors

- finally, we want to check the bayes facotor to determine what are the chances that Pfizer vaccine is more than 50% effective
- we define a conservative prior assuming that Pfizer vaccine is 50% effective with a standard deviation of 15% ➜ we assume a Normal distribution with `mean = 50%` and `sd = 15%`
- we then compute the odds that with our prior and posterior the Vaccine is more that 50% effective
- we use the `bayestestR` package:
  `https://github.com/easystats/bayestestR`
- the odds given our data are more than 400,000:1 that the vaccine is more than 50% effective, a very strong evidence

```
prior <- bayestestR::distribution_normal(60000, mean = 50, sd = 15)

bayestestR::bayesfactor_parameters(pfizer_res$diff_rate,
                                         prior,
                        direction = "two-sided", null = 50)

#> Bayes Factor (Savage-Dickey density ratio)
#> BF
#> ---------
#> 6.425e+05

* Evidence Against The Null: [50]
```

The `brms` package

References:

- CRANL: `https://cran.r-project.org/web/packages/brms/index.html`
- GitHub: `https://github.com/paul-buerkner/brms`

From the `brms` GitHub pages

The brms package provides an interface to fit Bayesian generalized (non-)linear multivariate multilevel models using Stan, which is a C++ package for performing full Bayesian inference (see `https://mc-stan.org/`). Formula syntax is very similar to that of the package lme4 to provide a familiar and simple interface for performing regression analyses. A wide range of response distributions are supported, allowing users to fit – among others – linear, robust linear, count data, survival, response times, ordinal, zero-inflated, and even self-defined mixture models all in a multilevel context. Further modeling options include non-linear and smooth terms, auto-correlation structures, censored data, missing value imputation, and quite a few more. In addition, all parameters of the response distribution can be predicted in order to perform distributional regression. Multivariate models (i.e., models with multiple response variables) can be fit, as well. Prior specifications are flexible and explicitly encourage users to apply prior distributions that actually reflect their beliefs. Model fit can easily be assessed and compared with posterior predictive checks, cross-validation, and Bayes factors.

brms example: Moderna age data

- Moderna released their RCT data grouped by age, considering older (age > 65 yr) and younger (≤ 65 yr) patients

Treatment	Age	Positive	Negative	Total
Placebo	≤ 65	79	8271	8350
Placebo	> 65	11	4494	4505
Vaccine	≤ 65	1	8293	8294
vaccine	> 65	4	4497	4501

- we treat COVID-19 outcomes as simple bernoulli events (again)
- brm allows us to specify the outcomes aggregated: i.e. the number of those that become infected according to their conditions (treatment and age)

```
age <- c(rep("lt65", 8350), rep("Older", 4505),
         rep("lt65", 8294), rep("Older", 4501))
treatment <- c(rep("Placebo", 8350), rep("Placebo", 4505),
         rep("Vaccine", 8294), rep("Vaccine", 4501))
tested <- c(rep("Pos", 79), rep("Neg", 8271),
            rep("Pos", 11), rep("Neg", 4494),
            rep("Pos", 1),  rep("Neg", 8293),
            rep("Pos", 4),  rep("Neg", 4497))

moderna_tb <- tibble(age = age, tested = tested,
                     treatment = treatment)
```

brms example: Moderna age data

- running brms

```
moderna_bf <- brm(data = moderna_tb,
                  family = bernoulli(link = logit),
                  tested ~ age + treatment + age:treatment,
                  iter = 12500, warmup = 500, chains = 4, cores = 12,
                  control = list(adapt_delta = .99, max_treedepth = 12),
                  seed = 9,
                  file = "moderna_long")

summary(moderna_bf)
 Family: bernoulli
  Links: mu = logit
Formula: tested ~ age + treatment + age:treatment
   Data: moderna_tb (Number of observations: 25650)
Samples: 4 chains, each with iter = 12500; warmup = 500; thin = 1;
         total post-warmup samples = 48000

Population-Level Effects:
                         Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
Intercept                   -4.66      0.11    -4.88    -4.44 1.00    39093
ageOlder                    -1.39      0.33    -2.07    -0.79 1.00    16404
treatmentVaccine            -4.74      1.17    -7.51    -2.95 1.00    10026
ageOlder:treatmentVaccine    3.65      1.32     1.42     6.66 1.00    10513
                           Tail_ESS
Intercept                     30740
ageOlder                      18439
treatmentVaccine               9939
ageOlder:treatmentVaccine     10761
```

a note on the logistic model

- a logistic model (logit) is used to model the probability of binary dependent variables
- let's have a model with one predictor x and one binary response variable y. y can be 0 or 1 and follows a Bernoulli probability
- we assume a linear relationship between the probability of success $p = P(y = 1)$ and the log-odds l:

$$l = \log \frac{p}{1-p} = a + b \cdot x$$

- the inverse transform

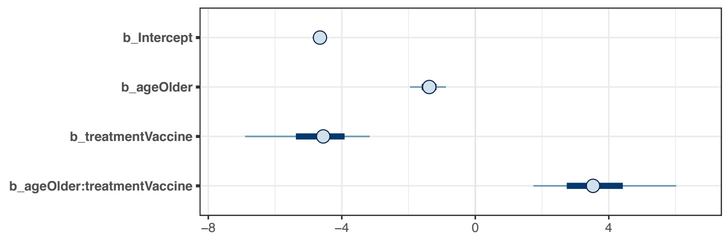
$$\frac{p}{1-p} = \exp a + b \cdot x$$

- gives p through of a Sigmoid function

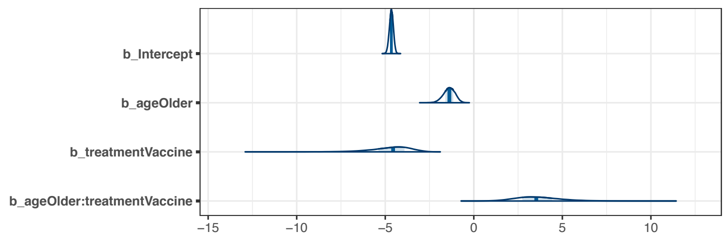
$$p = \Sigma(a + b \cdot x) = \frac{1}{1 - \exp a + b \cdot x}$$

brms example: Moderna age data

```
mcmc_plot(moderna_bf)
```



```
mcmc_plot(moderna_bf, type='areas')
```



```
bayestestR::describe_posterior(
                moderna_bf,
                ci=0.95,
                test=c("p_direction"),
                centrality="MAP")

Summary of Posterior Distribution

Parameter                  |   MAP |          95% CI |     pd |  Rhat |
ESS
------------------------------------------------------------------------
(Intercept)                | -4.66 | [-4.88, -4.43] |   100% | 1.000 | 38859.00
ageOlder                   | -1.38 | [-2.04, -0.76] |   100% | 1.000 | 16170.00
treatmentVaccine           | -4.25 | [-7.07, -2.71] |   100% | 1.001 |  8919.00
ageOlder:treatVaccine      |  3.21 | [ 1.22,   6.35] | 99.97% | 1.001 |  9677.00
```