Surveillance analysis Tool for Outcome-based Comparison of the confidence of FREEdom

The STOC free model

Estimation and implementation

Aurélien Madouasse

&

the STOC free consortium

https://www.stocfree.eu/

June 17, 2021

Table of Contents

Table of Contents

Data, hypotheses and parameters

- Data: test results, risk factors
- What we need to know: probability of infection on the current month
- What we know (more or less): test characteristics, characteristics of infection dynamics ...
- The modelling framework needs to be able to predict herd level probabilities of infection from test results and knowledge about test characteristics
- Chosen approach: Bayesian inference

Bayes' theorem

- What is a conditional probability?
  - Probability of an event given that another event has already happened

Sensitivity = probability of a positive test result (T^+) given that (|) an individual is diseased (D^+)

$$Se = p(T^+|D^+)$$

Positive predictive value = probability that an individual is diseased given a positive test result

$$PPV = p(D^+|T^+)$$

Bayes' theorem

- What is Bayes' theorem?
  - A simple formula that relates $p(B|A)$ to $p(A|B)$

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)}$$

Bayes' theorem

- Bayes' theorem applied to determining the probability of disease given a positive test result
  - Usually we know test sensitivity, and we would like to know the probability that disease is present given test result

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+)}$$

- $p(D^+|T^+)$: Positive predictive value
- $p(T^+|D+)$: Test sensitivity
- $p(D^+)$: Disease prevalence
- $p(T^+)$: Probability of a positive test

Bayes' theorem

- Bayes' theorem applied to determining the probability of disease given a positive test result

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+)}$$

Bayes' theorem

- Bayes' theorem applied to determining the probability of disease given a positive test result

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+)}$$

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+|D+)p(D^+) + p(T^+|D^-)p(D^-)}$$

Bayes' theorem

- Bayes' theorem applied to determining the probability of disease given a positive test result

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+)}$$

$$p(D^+|T^+) = \frac{p(T^+|D+)p(D^+)}{p(T^+|D+)p(D^+) + p(T^+|D^-)p(D^-)}$$

$$p(D^+|T^+) = \frac{Se\pi}{Se\pi + (1 - Sp)(1 - \pi)}$$

Bayes' theorem

Bayes' theorem applied to statistical inference

- We have some data (y) and a model, we would like to know what is the probability of the model parameter (θ) values given these data

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

$p(\theta|y)$ Probability of parameters given data $\rightarrow$ **Posterior distribution**

$p(y|\theta)$ Probability of data given parameters $\rightarrow$ **Likelihood function**

$p(\theta)$ Parameter prior distributions$\rightarrow$ **priors**

$p(y)$ **Normalising constant**

Bayes' theorem

Bayes' theorem applied to statistical inference

- Bayesian inference is a way to estimate model parameters incorporating:
    - data
    - prior knowledge/hypotheses about the model parameters

Bayes' theorem

Bayes' theorem applied to statistical inference

- The normalising constant $p(y)$:
  - is an integral that cannot be readily computed, except in simple cases
  - makes the area under the posterior density curve sum to 1
  - is a constant

$$p(y) = \int p(y|\theta)p(\theta)d\theta$$

Bayes' theorem

Bayes' theorem applied to statistical inference

- Because in most cases the normalising constant cannot be computed, we need estimation methods that do no need to compute it for the estimation of the full posterior density

$$p(\theta|y) \propto p(y|\theta)p(\theta)$$

- Solution: draw many samples from likelihood x prior distribution using Markov Chain Monte Carlo

Markov Chain Monte Carlo

- **Monte Carlo**: draw random samples (θ) from statistical distributions
- **Markov Chain**: the next random values drawn depend on the values of the current ones

$$p(\theta|y) \propto p(y|\theta)p(\theta)$$

Markov Chain Monte Carlo

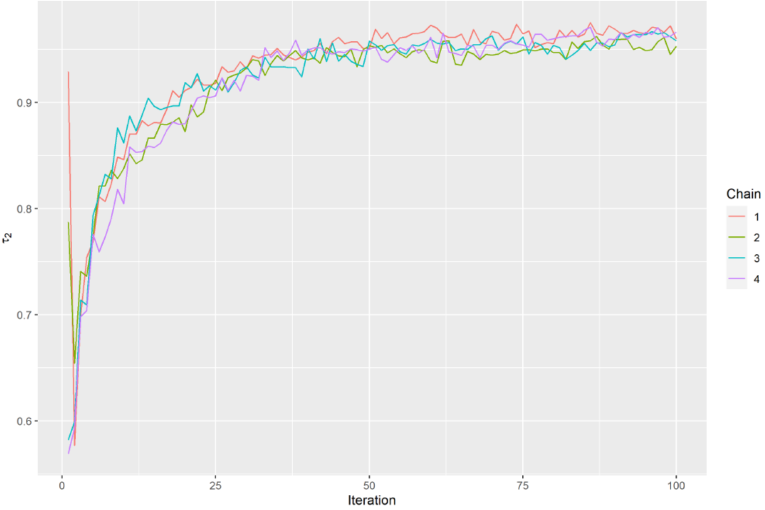
Principles of MCMC algorithms

- Start with some random initial values ($t = 1$)
- Use values at current iteration to sample values at next iteration (Markovian transition)
- The Markov Chain is constructed in such a way that it moves towards the target posterior probability distribution
- There is no way to be absolutely sure that the samples come from the target distribution
    - First iterations discarded $\Rightarrow$ burn in or warmup
    - Different simulations are run in parallel $\Rightarrow$ chains

Markov Chain Monte Carlo

Convergence

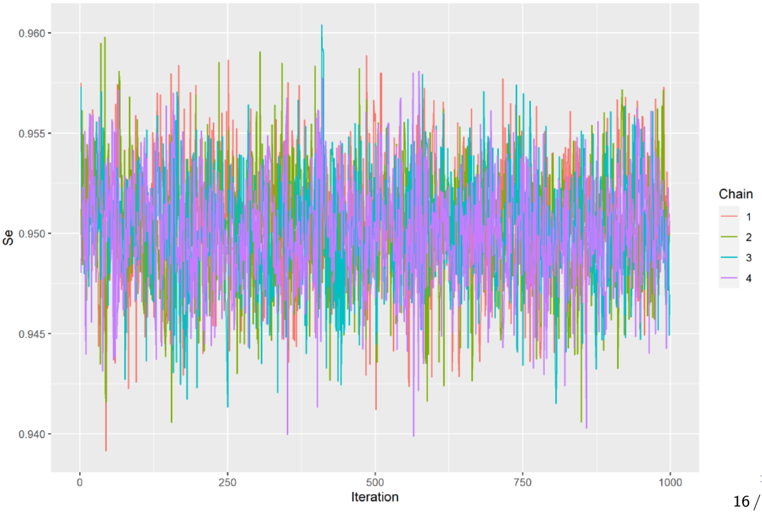
- Moving towards the target distribution



Markov Chain Monte Carlo

Convergence

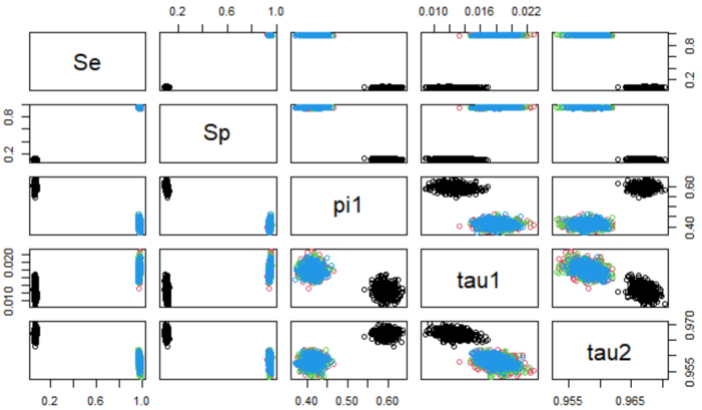
- All chains should converge to the same distribution → traceplot



Markov Chain Monte Carlo

Convergence

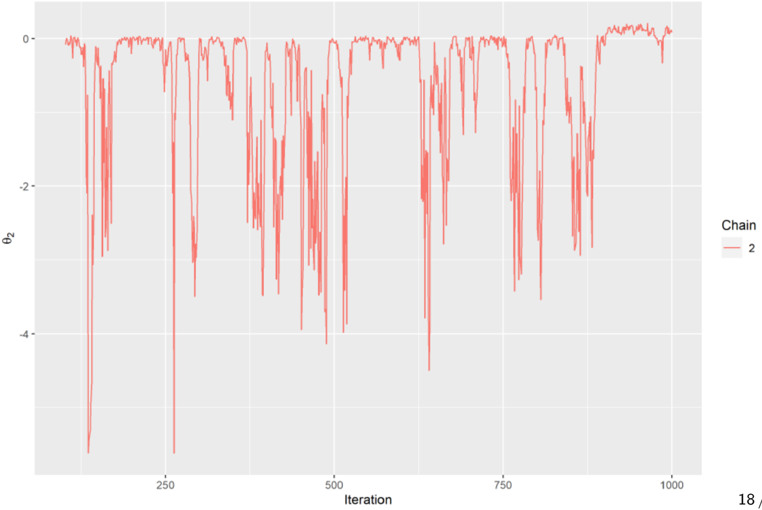
- Different chains can converge to different distributions
  - Model with 5 parameters / each color is a chain



Markov Chain Monte Carlo

Autocorrelation

- Autocorrelation: within a chain, high correlation between consecutive MCMC samples



Markov Chain Monte Carlo

in pratcice

- Run several chains (> 2): allows checking that the samples obtained do not come from different distributions = convergence (traceplots , Gelman Rubin statistic)
- Initialise each chain with different values
- Discard the first n iterations = burn in, warmup
- If autocorrelation, use 1 out of k iterations (depending on autocorrelation) = thinning

Markov Chain Monte Carlo

Algorithms

- There exist many different MCMC algorithms:
  - Metropolis: first invented (1953)

    See an introduction here: Ben Lambert - An introduction to the Random Walk Metropolis algorithm
  - Metropolis Hastings
  - Gibbs sampling: BUGS, WinBUGS, JAGS
  - Hamiltonian Monte Carlo: Stan

Markov Chain Monte Carlo

Gibbs sampling

- First widely used algorithm for Bayesian inference
  - Not possible before the 1990s because computation intensive
  - First implementations:
    - BUGS = Bayesian Inference Using Gibbs Sampling
    - WinBUGS , OpenBUGS
  - Most recent implementations
    - JAGS: Just Another Gibbs Sampler
    - MultiBUGS

  $\Rightarrow$ Same principles, more efficient

Markov Chain Monte Carlo

Gibbs sampling

- All the programmes use the same language to code statistical models
- Easy to write the code from model specification
- Straightforward to translate the STOC free model equations into code

Markov Chain Monte Carlo

Hamiltonian Monte Carlo

- Implemented in Stan
- Much more efficient than Gibbs sampling
  - Exploration of the posterior distribution much more efficient
  - Less autocorrelation $\rightarrow$ requires less iterations
- Does not support latent discrete parameters
  - Not possible to code the STOC free model as simply as in JAGS

Markov Chain Monte Carlo

- For a visual comparison of different MCMC algorithms, see: The Markov-chain Monte Carlo Interactive Gallery by Chi-Feng

Table of Contents

JAGS model

- Easy to go from model equations to JAGS code
- On the following slides:
  - Simplified version in which test results assumed available for all months
  - The *real* model allows for missing test results with a complicated system of loops. Same idea but harder to read
  - When no test available, the dynamics drive status evolution

JAGS model

First status

```
model{
  ## loop over all herds
  ## t1 is the vector of indices for first month of test in each herd
  ## t2 is the vector of indices for second month of test in each herd
  ## tf is the vector of indices for last month of test in each herd
  for(i in 1:N_herds){

    ### First monthly status of each herd
    ## probability of being latent status positive for herd i at t1
    logit_pi1[i] ~ dnorm(logit_pi1_mean, logit_pi1_prec)

    ## latent status for herd i at time = 1
    Status[t1[i]] ~ dbern(ilogit(logit_pi1[i]))

    ## probability of being test positive given herd status
    p_test_pos[t1[i]] <- Se * Status[t1[i]] +
                         (1 - Sp) * (1 - Status[t1[i]])

    ## test result associated with first Status => data
    test_res[t1[i]] ~ dbern(p_test_pos[t1[i]])
...
```

JAGS model

Statuses 2 to last - 1

```
### Statuses 2 to  1 minus last
for(t in (t1[i] + 1):(tf[i] - 1)){

  # probability of new infection
  # logistic regression
  logit(tau1[t]) <- inprod(risk_factors[t,], theta)

  ## probability of being status positive given previous status,
  ## tau1 and tau2
  pi[t] <- (1 - Status[t - 1]) * tau1[t] +
           Status[t - 1] * tau2

  ## herd status at time t
  Status[t] ~ dbern(pi[t])

  ## probability of test positive at time t
  p_test_pos[t] <- Se * Status[t] + (1 - Sp) * (1 - Status[t])

  ## test result at time t  => data
  test_res[t] ~ dbern(p_test_pos[t])

}
```

JAGS model

Predicted statuses

```
  # probability of new infection
  logit(tau1[tf[i]]) <- inprod(risk_factors[tf[i],], theta)

  ## Predicted probability of infection for herd i on last month
  pi[tf[i]] <- tau1 * (1 - Status[tf[i] - 1]) +
               tau2 * Status[tf[i] - 1]
  # probability of infection updated with test result
  predicted_proba[tf[i]] <- test_res[tf[i]] * (
    Se * pi[tf[i]] / (Se * pi[tf[i]] + (1 - Sp) * (1 - pi[tf[i]]))
  ) + (1 - test_res[tf[i]]) * (
      (1 - Se) * test_res[tf[i]] /
        ((1 - Se) * pi[tf[i]]  + Sp * (1 - pi[tf[i]] ) )
    )

}
```

JAGS model

Priors

```
### Priors
## test characteristics
Se ~ dbeta(Se_beta_a, Se_beta_b)
Sp ~ dbeta(Sp_beta_a, Sp_beta_b)

## Status dynamics - sampling on the logit scale
logit_tau2 ~ dnorm(logit_tau2_mean, logit_tau2_prec)

## logit back to the probability scale
tau2 <- ilogit(logit_tau2)

## Logistic regression coefficients
for(i_rf in 1:n_risk_factors){

  theta[i_rf] ~ dnorm(theta_norm_mean[i_rf], theta_norm_prec[i_rf])

}

}
```

Stan model

- Stan implements Hamiltonian Monte Carlo which is expected to be more efficient at sampling from the full posterior distribution
- Stan does not support latent discrete parameters
- Translation of the model's equations not as easy as with JAGS
- Various HMM implementations in Stan described in a tutorial by Damiano et al. (2017)

Stan model

- Forward algorithm adapted from the tutorial
- Same model as the JAGS version, but estimation performed in a different way

Stan model

Declaration of variables

```
data{

  int<lower=1> n_herds;
  int<lower=1> herds_t1[n_herds];
  int<lower=1> herds_t2[n_herds];
  int<lower=1> herds_T[n_herds];
  int<lower=1> N;
  int<lower=0, upper=3> test_res[N];
  real<lower = 0> Se_beta_a;
  real<lower = 0> Se_beta_b;
  real<lower = 0> Sp_beta_a;
  real<lower = 0> Sp_beta_b;
  real logit_pi1_mean;
  real logit_pi1_sd;
  real logit_tau2_mean;
  real logit_tau2_sd;
  int<lower = 0> n_risk_factors;
  real theta_norm_mean[n_risk_factors];
  real theta_norm_sd[n_risk_factors];
  matrix[N, n_risk_factors] risk_factors;

}
```

Stan model

Parameters

```
parameters{

  real<lower = 0, upper = 1> Se;
  real<lower = 0, upper = 1> Sp;
  real<lower = 0, upper = 1> pi1;
  real<lower = 0, upper = 1> tau2;
  vector[n_risk_factors] theta;

}
```

Stan model

Transformed parameters

```
transformed parameters{

  // logalpha needs to be accessible to other blocks
  matrix[N, 2] logalpha;

  {

    // accumulator used at each time step
    real tau1[N];
    real accumulator[2];

    // logistic regression for tau1
  for(n in 1:N){

  tau1[n] = inv_logit(risk_factors[n,] * theta);

  }
```

Stan model

First status

```
// looping over all herds
for(h in 1:n_herds){

  // first test in sequence
  // negative status
  logalpha[herds_t1[h], 1] = log(1 - pi1) +
                 bernoulli_lpmf(test_res[herds_t1[h]] | 1 - Sp);
  // positive status
  logalpha[herds_t1[h], 2] = log(pi1) +
                 bernoulli_lpmf(test_res[herds_t1[h]] | Se);
```

Stan model

Status transition / no test result

```
// tests 2 in T in sequence
  for(t in herds_t2[h]:herds_T[h]){

// Missing test result
     if(test_res[t] == 3){

// transition from status negative to status negative (j = 1; i = 1)
     accumulator[1] = logalpha[t-1, 1] + log(1 - tau1[t]);
// transition from status positive to status negative (j = 1; i = 1)
     accumulator[2] = logalpha[t-1, 2] + log(1 - tau2);

     logalpha[t, 1] = log_sum_exp(accumulator);

// transition from status negative to status negative (j = 1; i = 1)
     accumulator[1] = logalpha[t-1, 1] + log(tau1[t]);
// transition from status positive to status positive (j = 1; i = 1)
     accumulator[2] = logalpha[t-1, 2] + log(tau2);

     logalpha[t, 2] = log_sum_exp(accumulator);

    } else {
```

Stan model

Status transition / test result

```
// transition from status negative to status negative (j = 1; i = 1)
    accumulator[1] = logalpha[t-1, 1] +
                     log(1 - tau1[t]) +
                     bernoulli_lpmf(test_res[t] | 1 - Sp);
// transition from status positive to status negative (j = 1; i = 1)
    accumulator[2] = logalpha[t-1, 2] +
                     log(1 - tau2) +
                     bernoulli_lpmf(test_res[t] | 1 - Sp);

    logalpha[t, 1] = log_sum_exp(accumulator);
```

Stan model

Status transition / test result

```
// transition from status negative to status negative (j = 1; i = 1)
   accumulator[1] = logalpha[t-1, 1] +
                    log(tau1[t]) +
                    bernoulli_lpmf(test_res[t] | Se);
// transition from status positive to status positive (j = 1; i = 1)
   accumulator[2] = logalpha[t-1, 2] +
                    log(tau2) +
                    bernoulli_lpmf(test_res[t] | Se);

   logalpha[t, 2] = log_sum_exp(accumulator);

       } // if

     } // time sequence loop

    } // herd loop

  } //local

} // end of block
```

Stan model

Priors and likelihood

```
model{

// priors for test characteristics
     Se ~ beta(Se_beta_a, Se_beta_b);
     Sp ~ beta(Sp_beta_a, Sp_beta_b);

// priors for status dynamics
  logit(pi1) ~ normal(logit_pi1_mean, logit_pi1_sd);
  logit(tau2) ~ normal(logit_tau2_mean, logit_tau2_sd);

// priors for the logistic regression coefficients
  for(k in 1:n_risk_factors){

  theta[k] ~ normal(theta_norm_mean[k], theta_norm_sd[k]);

  }

// update based only on last logalpha of each herd
  for(i in 1:n_herds)
    target += log_sum_exp(logalpha[herds_T[i]]);

}
```

Stan model

Predicted status

```
generated quantities{

  // variable in which predictions are stored
  real pred[n_herds];

  {
    matrix[n_herds, 2] alpha;

  // loop in which the probabilities of infection are predicted
    for(i in 1:n_herds){
      alpha[i] = softmax(logalpha[herds_T[i],]')';
      pred[i] = alpha[i, 2];
    }
  }
}
```

Comparison of the JAGS and Stan implementations

- The JAGS and Stan implementations of the model were compared using data collected as part of BVDV control programme in France
  - Work under review with PCI Animal Science, available as a pre-print.
- The Stan implementation:
  - gives the same parameter estimates
  - is much faster
  - converges much better
  - returns predicted probabilities of infection that are easier to interpret

The `STOCfree` R package

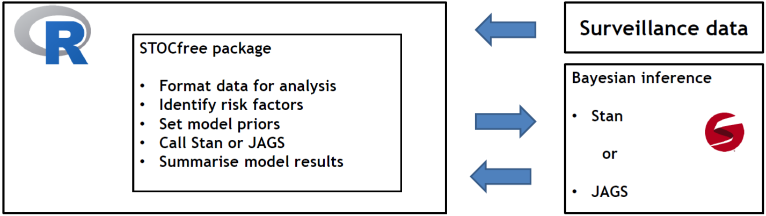
What is an R package?

- Programming environment for data manipulation and analysis
- Widely used
- Free

package

- Set of functions gathered to perform specific tasks
- Users install a package and can use the functions they contain
- Packages are installed from the web (CRAN, GitHub. . .)

The STOCfree R package

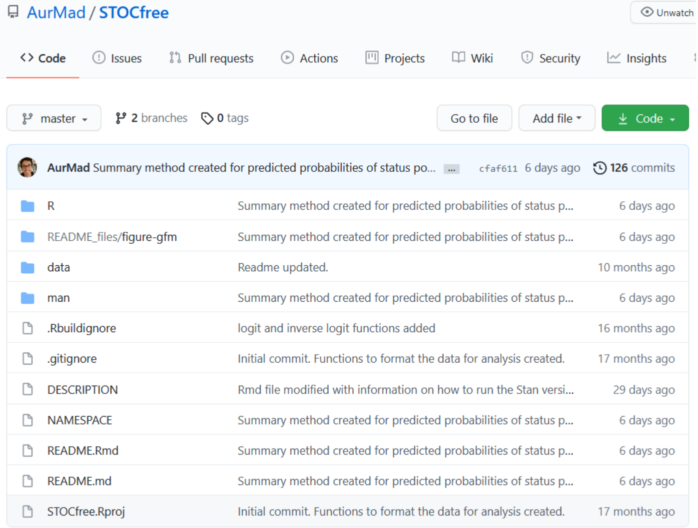


The STOCfree R package on Github

- The package is hosted on Github
  https://github.com/AurMad/STOCfree
- Github is a server hosting:
  - The package code
  - The package documentation
  - The history of development and different package versions, using the Git versioning programme

The STOCfree R package on Github

- All the code is in the R folder



The STOCfree R package on Github

- The documentation is at the bottom of the page

README.md

STOCfree: prediction of probabilities of freedom from infection from longitudinal data

Overview

The aim of the `STOCfree` package is to predict herd level probabilities of freedom from infection from

Thank you for your attention

http://www.stocfree.eu/

This study was awarded a grant by EFSA and was co-financed by public organisations in the countries participating in the study.