



"Rather than learning how to solve that, shouldn't we be learning how to operate software that can solve that problem?"

Stan: orodje za uporabno Bayesovo statistiko

Erik Štrumbelj
2018

Glavne teme

- **Orodje Stan:**

- Osnovna ideja za orodjem Stan.
- Kratek opis.
- Vmesniki in napredne funkcionalnosti.

- **Programski jezik Stan:**

- Sintaksa: bloki, podatkovni tipi, vgrajene funkcije.
- Stan program = opis modela = navodila za izračun $p(\theta|y)$.

- **Praktični primeri:**

- Poisson-Gamma, Bernoulli-Beta, Normal-InvGamma...
- Diagnosticiranje MCMC: traceplot, ESS, SE_{MCMC} .
- Generiranje novih primerov iz $p(y_{new}|y)$ in drugih izpeljanih vrednosti.

Standardni koraki Bayesove statistike

Podatki:	y_1, \dots, y_n	$y_i \in \{0, 1, 2, 3, \dots\}'$
Model (verjetje):	$y_1, y_2, \dots, y_n \theta \sim_{\text{iid}} \text{Poisson}(\theta)$	
Apriorna porazdelitev:	$\theta \sim \text{Gamma}(a, b)$	

————— konec statistike, začne se računanje —————

Aposteriorna porazdelitev: $p(\theta|y) = \frac{p(\theta) \prod_{i=1}^n p(y_i|\theta)}{p(y)} = \frac{b^a}{\Gamma(a)} \theta^{a+\sum y_i-1} e^{-(b+n)\theta}$
Sklepanje iz aposteriorne porazdelitve: $\theta|y \sim \text{Gamma}(a + \sum y_i, b + n)$

(včasih je $p(\theta|y)$ standardna porazdelitev, večinoma sklepamo z aproksimacijo, predvsem MCMC)

————— konec računanja, začne se interpretacija —————

Preverjanje kvalitete modela, verjetnosti hipotez, porazdelitve parametrov...

Splošni pristopi k sklepanju iz modelov (računanju)

Splošnonamensko orodje za Bayesovo statistiko mora biti zmožno sklepati iz poljubnega modela (ali vsaj zelo širokega razreda modelov).

Model in apriorne enolično opredelijo aposteriorno porazdelitev:

$$p(\theta|y) = \frac{p(\theta) \prod_{i=1}^n p(y_i|\theta)}{p(y)} \propto \text{dgamma}(\theta|a, b) \prod_{i=1}^n \text{dpois}(y_i|\theta)$$

Običajno delamo z logaritmom aposteriorne (numerična stabilnost):

$$\log p(\theta|y) = \log(\text{dgamma}(\theta|a, b)) + \sum_{i=1}^n \log(\text{dpois}(y_i|\theta)) + C$$

Postopek, ki opis modela prevede na simboličen zapis izračuna (log)aposteriorne je enostavno avtomatizirati. **Zakaj?**

Splošni pristopi k sklepanju iz modelov (računanju)

Ko znamo izračunati (ovrednotiti) vrednost (log)aposteriorne v poljubni točki prostora parametrov, znamo vzorčiti iz nje (Metropolis-Hastings)!

Kljub temu ni orodja, ki bi temeljilo na algoritmu M-H. **Zakaj?**

Splošni pristopi k sklepanju iz modelov (računanju)

Ko znamo izračunati (ovrednotiti) vrednost (log)aposteriorne v poljubni točki prostora parametrov, znamo vzorčiti iz nje (Metropolis-Hastings)!

Kljub temu ni orodja, ki bi temeljilo na algoritmu M-H. **Zakaj?**

Parametrov M-H (porazdelitve kandidatov) ne znamo prilagajati na način, ki je dovolj učinkovit in hkrati dovolj enostaven, da je splošno uporaben.

Dve generaciji orodij za Bayesovo statistiko:

- Prva generacija (1989-...): temelji na vzorčevalniku Gibbs (BUGS, WinBUGS, OpenBUGS, JAGS).
- Druga generacije (2011-...): temelji na HMC - Hamiltonian Monte Carlo (Stan,...).

Splošni pristopi k sklepanju iz modelov (računanju)

Hamiltonski Monte Carlo:

(včasih tudi hibridni Monte Carlo, ker združuje hamiltonske sisteme in MCMC)

- Bolj učinkovit, ker upošteva več informacij o funkciji (gradient).
- Algoritem znan že od leta 1987, v splošni uporabi šele danes.
- Glavna težava v tem, kako samodejno računati gradient!
- Simbolično in numerično odvajanje počasna, nestabilna.
- Glavni preboj: avtomatsko odvajanje (Automatic Differentiation):
 - Definiramo odvod funkcij v našem jeziku, uporabimo pravila odvajanja.
 - Kot da bi kodo funkcije prevedli v kodo za izračun gradienta.
 - Natančnost omejena z natančnostjo računalnika.

Osnovni gradniki orodja Stan



Orodje Stan je sestavljeno iz

- 1 programskega jezika Stan za opisovanje modela in apriornih,
- 2 algoritma za prevod opisa v izračun aposteriorne (gradienta) in
- 3 algoritma Hamiltonian Monte Carlo za vzorčenje iz aposteriorne.

<http://mc-stan.org/>

Povzetek:

- Orodje/programski jezik za bayesovsko statistično sklepanje,
- uporablja Hamiltonian Monte Carlo (različica M-H),
- napisan v C++ (Win, Mac, Linux),
- jezik ima C-jevsko sintakso s pridihom R-ja,
- vmesniki za R, Python, MATLAB, Julia, Stata, Mathematica, Scala...
- open source (BSD, PLv3),
- zelo dobra dokumentacija (priročnik, študije,...),
- stalen razvoj, aktivna skupnost.

Stan - dodatne funkcionalnosti

Več različnih aproksimacijskih algoritmov:

- Privzet vzorčevalnik je NUTS (No-U-Turn Sampler, različica HMC).
- HMC.
- Strukturne aproksimacije (Variational Inference, Laplace).
- Klasična optimizacija (L-BFGS, BFGS).

Uporabniku prijaznejši vmesniki, orodja (v R):

- RStanArm: bayesovski ekvivalenti `lm()`, `glm()`, `polr()`...
- ShinyStan: Interaktivni povzetki in napredna diagnostika MCMC.
- bayesplot: Vizualizacija rezultatov MCMC.
- loo: Ocenjevanje in primerjava kvalitete modelov na podlagi MCMC.

Sintaksa programskih jezikov

Naš prvi cilj bo, da model Poisson-Gamma napišemo v Stan-u!

Da bi razumeli jezik, moramo najprej razumeti njegovo sintakso.

Ponovimo (kontekstno neodvisne) gramatike:

```
STAVEK ::= ?osebek povedek ?predmet
povedek ::= 'je' | 'vozi' | 'bere'
osebek  ::= 'Erik ' | 'pes ' | 'avto '
predmet ::= ' klobaso' | ' tovor' | ' knjigo'
```

(::= lahko zamenjamo z; | ali; ? lahko je, lahko ni; * poljubno mnogo)

Gramatika je sestavljena iz nekončnih simbolov (povedek, osebek...), začetnega simbola (STAVEK), končnih simbolov ('je', 'pes ', ' tovor'...) in množice predvedb, ki nam povedo, kaj lahko izpeljemo iz začetnega simbola. Gramatika definira jezik (množico zaporedij končnih simbolov, ki jih lahko izpeljemo).

Sintaktična gramatika programskega jezika Stan

O sintaksi se lahko učimo iz primerov, a gramatika je popoln in nedvoumen vir informacij, kaj se šteje za sintaktično pravilen program.

Začetni simbol je `program`:

```
program ::= ?functions ?data ?tdata ?params ?tparams model ?generated
```

Vsak program v Stan-u je sestavljen iz sedmih blokov. Blok `model` je obvezen, ostali so izbirni.

Najkrajši sintaktično pravilen program v Stan-u je?

Sedem blokov programa

Vsak blok se začne z imenom, sledi pa poljubno mnogo deklaracij funkcij, spremenljivk, stavkov (odvisno od bloka):

```
functions ::= 'functions' function_decls
data ::= 'data' var_decls
tdata ::= 'transformed data' var_decls_statements
params ::= 'parameters' var_decls
tparams ::= 'transformed parameters' var_decls_statements
model ::= 'model' statement
generated ::= 'generated quantities' var_decls_statements

function_decls ::= '{' function_decl* '}'
var_decls ::= '{' var_decl* '}'
var_decls_statements ::= '{' var_decl* statement* '}'
```

Zaenkrat se bomo osredotočili na tri najbolj pomembne bloke: data, parameters in model.

Bloka data in parameters

Bloka data in parameters vsebujeta le deklaracije spremenljivk:

```
data ::= 'data' var_decls
params ::= 'parameters' var_decls

var_decls ::= '{' var_decl* '}'
```

Čeprav sta bloka sintaktično enaka, se razlikujeta v pomenu:

- V bloku data deklariramo podatke. Vrednosti teh spremenljivk bodisi mora podati upravnik bodisi jih določimo v bloku. Blok služi opisu podatkov in apriornih konstant.
- V bloku parameters deklariramo parametre modela.

Deklariranje spremenljivk

Spremenljivko deklariramo z njenim tipom in imenom. Lahko dodamo tudi dimenzionalnost (če želimo polje oz. array) in ji določimo vrednost.

```
var_decl ::= var_type variable ?dims ?('=' expression) ';'
dims ::= '[' expressions ']'
variable ::= identifier
identifier ::= [a-zA-Z] [a-zA-Z0-9_]*
```

Imena spremenljivk se začnejo s črko in vsebujejo le črke, številke in podčrtaj.

Podatkovni tipi

Stan podpira standardne tipa `int` in `real` ter več različnih vektorskih oz. matričnih tipov, pri katerih določimo dimenzionalnostⁱ.

```
var_type ::= 'int' range_constraint | 'real' range_constraint
| 'vector' range_constraint '[' expression ']'
| 'ordered' '[' expression ']' | 'positive_ordered' '[' expression ']'
| 'simplex' '[' expression ']'
| 'unit_vector' '[' expression ']'
| 'row_vector' range_constraint '[' expression ']'
| 'matrix' range_constraint '[' expression ',' expression ']'
| 'cholesky_factor_corr' '[' expression ']'
| 'cholesky_factor_cov' '[' expression ?(',' expression) ']'
| 'corr_matrix' '[' expression ']' | 'cov_matrix' '[' expression ']'
```

Zaenkrat bomo shajali z `int` in `real`.

ⁱPozor, to ni enako kot polje!

Določanje zgornje in spodnje meje

Stan omogoča, da dodatno omejimo možne vrednosti spremenljivk:

```
range_constraint ::= ?('<' range '>')  
range ::= 'lower' '=' expression ',' 'upper' = expression  
| 'lower' '=' expression  
| 'upper' '=' expression
```

V bloku data služijo le preverjanju podatkov - kršenje izzove napako.

V bloku parameters Stan-u sporočajo, kje je veljaven prostor parametrov. Stan vzorči/optimizira le na tem prostoru. Omejitve so zelo pomembne pri parametrih, ki tudi sicer omejeni, npr. varianca ali standardni odklon.

Blok model

Blok model je zaporedje deklaracij spremenljivk in stavkov. Služi opisu apriornih in verjetja. Osnovni stavek je lahko več različnih stvari:

```
statement ::= atomic_statement | nested_statement
atomic_statement ::= atomic_statement_body ';'
atomic_statement_body
::= lhs assignment_op expression
| expression '~' identifier '(' expressions ')' ?truncation
| function_literal '(' expressions ')'
| 'increment_log_prob' '(' expression ')' | 'target' '+=' expression
| 'break' | 'continue' | 'print' '(' (expression | string_literal)* ')'
| 'reject' '(' (expression | string_literal)* ')'
| 'return' expression | ''

assignment_op ::= '<-' | '=' | '+=' | '-=' | '*=' | '/=' | '.*=' | '/*='
string_literal ::= '"' char* '"'
truncation ::= 'T' '[' ?expression ',' ?expression ']'
lhs ::= identifier ?('[' indexes ']')
```

Blok model

Stan podpira tudi klasične kontrolne stavke if-then-else, while in for:

```
nested_statement
::= 'if' '(' expression ')' statement
    ('else' 'if' '(' expression ')' statement)*
    ?('else' statement)
    | 'while' '(' expression ')' statement
    | 'for' '(' identifier 'in' expression ':' expression ')' statement
    | '{' var_decl* statement+ '}'
```

Večina funkcij v stan je vektoriziranih, zato se zankam poskušamo izogniti (podobno, kot v R). Več o tem kasneje.

Opis apriorne porazdelitve

To, da menimo, da neka spremenljivka prihaja iz neke porazdelitve oz. na ta način izrazimo svoje apriorno mnenje, npr.

$$\theta \sim \text{Gamma}(a_0, b_0),$$


lahko v Stan-u naredimo na 2 načina:

Kot t. i. sampling statement (podobno naši notaciji!):

```
theta ~ gamma(a0, b0);
```

S prištevanjem gostote k vrednosti log-aposteriorne porazdelitve:ⁱⁱ

```
target += gamma_lpdf(theta | a0, b0);
```

ⁱⁱVčasih je bilo še `increment_log_prob(gamma_log(...))`; a se sedaj odsvetuje. 

Vgrajene funkcije

Osnovne matematične operacije:

`min, max, round, log, exp...`

En kup vektorskih, matričnih in drugih operacij.

Vse standardne (in malo manj standardne) porazdelitve:

- Log-gostote (npr. `normal_lpdf(x|mu,sigma)`),
- CDF (npr. `normal_cdf(x,mu,sigma)`),
- Vzorčenje (npr. `normal_rng(mu,sigma)`).

`bernoulli, poisson, normal, beta, gamma, inv_gamma...`

V skrajnem primeru pa lahko napišemo tudi svoje funkcije ali porazdelitve.

Opis verjetja

Ponavljajočo se naravo verjetja, npr.

$$y_1, \dots, y_n | \theta \sim_{\text{iid}} \text{Poisson}(\theta),$$

lahko v Stan-u opišemo na (vsaj) dva različna načina:

V zanki preko vseh primerov:

```
for (i in 1:n) {  
  y[i] ~ poisson(theta);  
}
```

Z izkoriščanjem vektorizacije argumentov:

```
y ~ poisson(theta);
```

(večina funkcij v Stan-u podpira vektorizacijo)

Primer (Model Poisson-Gamma)

```
data {  
  int n;  
  int y[n];  
  real a0;  
  real b0;  
}  
  
parameters {  
  real<lower = 0.0> lambda;  
}  
  
model {  
  lambda ~ gamma(a0, b0);  
  
  for (i in 1:n) {  
    y[i] ~ poisson(lambda);  
  }  
}
```


Sklepanje iz opisanega modela preko vmesnika

Namestite si vmesnik za vaš najljubši jezik: mc-stan.org/users/interfaces/index.html
Pri čemer pri tem predmetu privzemite, da je vaš najljubši programski jezik R.

R paket `rstan`, funkcija `stan()`:

```
stan(file, model_code = "", # pass model as file/string with model code...
      fit = NA,              # ...or as an already compiled model
      data = list(),         # the data, prior constants... passed to Stan
      pars = NA,             # output only these parameters, if specified
      chains = 4,            # number of MCMC chains to run
      iter = 2000,           # total number of samples
      warmup = floor(iter/2), # samples used for MCMC tuning
      init = "random",       # starting values
      seed = sample.int(.Machine$integer.max, 1),
      cores = getOption("mc.cores", 1L),
      ...,)
```

Funkcija vrne objekt tipa `stanfit`, ki vsebuje vzorce in še marsikaj.

Druge uporabne funkcije: `traceplot`, `plot`, `print`, `extract`...

Primer (Model Poisson-Gamma)

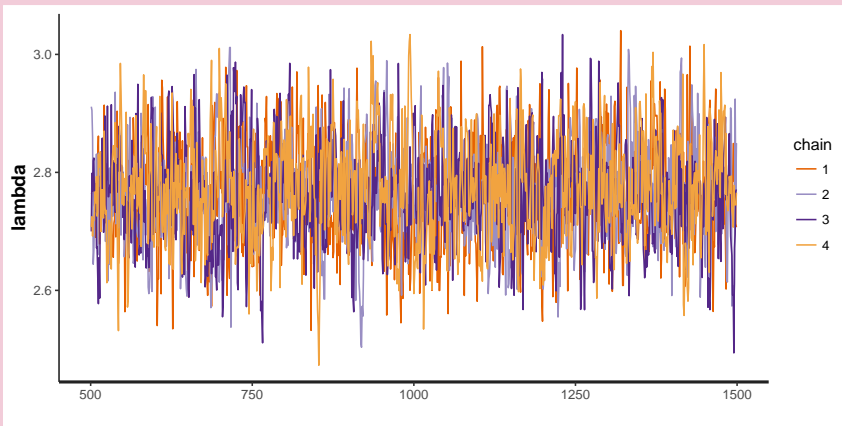
```
# Prepare dataset for Stan -----
stan_data <- list(y = y,
                  n = length(y),
                  a0 = 2,
                  b0 = 1)

# Sample -----
output <- stan(file = "../STANMODELS/poisson_gamma.stan",
               data = stan_data,
               iter = 1500, warmup = 500,
               chains = 4,
               seed = 0)

# Diagnostics & Posterior inference -----
traceplot(output)
plot(output)
print(output)
```

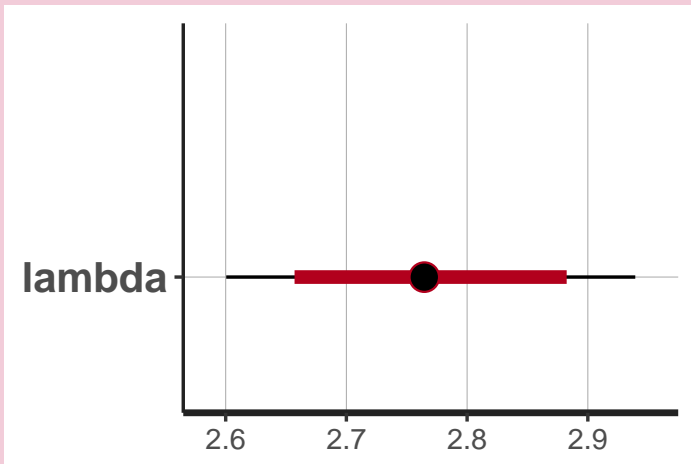
Primer (Model Poisson-Gamma)

```
> traceplot(output)
```



Primer (Model Poisson-Gamma)

```
> plot(output)
```



Primer (Model Poisson-Gamma)

```
> print(output)
```

Inference for Stan model: poisson_gamma.

4 chains, each with iter=1500; warmup=500; thin=1;

post-warmup draws per chain=1000, total post-warmup draws=4000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
lambda	2.77	0.00	0.09	2.60	2.71	2.76	2.82	2.94	1527	1
lp__	17.97	0.02	0.70	15.98	17.77	18.26	18.44	18.50	1637	1

Samples were drawn using NUTS(diag_e) at Mon Mar 26 05:46:21 2018.

For each parameter, `n_eff` is a crude measure of effective sample size, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat`=1).

Primer (Model Bernoulli-Beta)

Primer (Model Bernoulli-Beta)

```
data {  
  int n;  
  int y[n];  
  real a0;  
  real b0;  
}  
  
parameters {  
  real<lower = 0.0, upper = 1.0> p;  
}  
  
model {  
  p ~ beta(a0, b0);  
  
  for (i in 1:n) {  
    y[i] ~ bernoulli(p);  
  }  
}
```

Primer (Model Bernoulli-Beta)

```
# Toy data -----
set.seed(0)

coin1 <- sample(0:1, 100, prob = c(???, ???), rep = T)
coin2 <- sample(0:1, 20, prob = c(???, ???), rep = T)
coin3 <- sample(0:1, 10, prob = c(???, ???), rep = T)
coin4 <- sample(0:1, 4, prob = c(???, ???), rep = T)

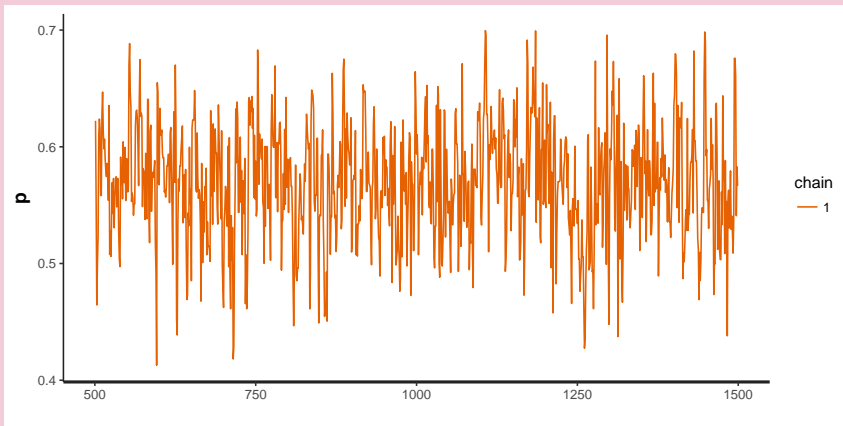
# 'ONE SAMPLE BINOMIAL TEST' MODEL -----
# Prepare dataset for Stan -----
stan_data <- list(y = coin1, n = length(coin1), a0 = 0.5, b0 = 0.5)

# Sample -----
output <- stan(file = "../STANMODELS/bernoulli_beta.stan", data = stan_data,
               iter = 1500, warmup = 500, chains = 1, seed = 0)

# Diagnostics & Posterior inference -----
traceplot(output)
print(output)
```


Primer (Model Bernoulli-Beta)

```
> traceplot(output)
```



Primer (Model Bernoulli-Beta)

```
> print(output)
```

Inference for Stan model: bernoulli_beta.

1 chains, each with iter=1500; warmup=500; thin=1;

post-warmup draws per chain=1000, total post-warmup draws=1000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
p	0.57	0.00	0.05	0.47	0.54	0.57	0.60	0.67	298	1
lp__	-69.54	0.03	0.72	-71.69	-69.75	-69.25	-69.08	-69.03	503	1

Samples were drawn using NUTS(diag_e) at Mon Mar 26 06:07:38 2018.

For each parameter, `n_eff` is a crude measure of effective sample size, and `Rhat` is the potential scale reduction factor on split chains (at convergence, `Rhat`=1).

Primer (Model Bernoulli-Beta: delo z vzorci)

```
> # Working just with the samples -----
> library(mcmcse)
> smp_p <- extract(output)$p
>
> # mean
> print(ess(smp_p))          # effective sample size
      se
981.4868
> print(mcse(smp_p))        # est. MCMC standard error
$est
[1] 0.5690762

$se
[1] 0.00157484

> print(acf(smp_p)$acf[1:6]) # autocorrelation (lag 0-5)
[1] 1.000000000 0.034055352 -0.008012901 -0.009085831 0.015316473 0.026158402
```

Primer (Model Poisson-Gamma: delo z vzorci)

```
> # P(p > 0.5)
> tmp <- ifelse(smp_p > 0.5, 1, 0)
> print(ess(tmp))          # effective sample size
      se
719.69
> print(mcse(tmp))         # est. MCMC standard error
$est
[1] 0.917

$se
[1] 0.01028888

> print(acf(tmp)$acf[1:6]) # autocorrelation (lag 0-5)
[1] 1.00000000 0.01450659 0.02755478 0.01432556 -0.02518106 0.02728324
```

Programersko-statistična naloga

Napišimo model, s katerim bomo lahko za več kovancev hkrati ocenili njihovo pričakovano vrednost in verjetnost, da je kovanec najboljši (ima najvišjo pričakovano vrednost).

Namig 1: `sort_indices_desc()`

Namig 2: blok `generated quantities`

Blok generated quantities

```
generated ::= 'generated quantities' var_decls_statements
```

```
var_decls_statements ::= '{' var_decl* statement* '}'
```

Namenjen je deklaraciji spremenljivk, ki so izpeljanke parametrov, niso del modela, a nas vseeno zanimajo:

- Izvede se enkrat v vsaki iteraciji (po vzorčenju).
- Deklariramo spremenljivke ter kako jih izpeljemo.
- Lahko tudi vzorčimo (uporabno za repliciranje množic za vizualno preverjanje modela).

Primer (Model Bernoulli-Beta za več skupin)

```
data {  
  int<lower=1> n;           // number of measurements  
  int<lower=1> k;           // number of coins  
  int<lower=0,upper=1> y[n]; // measurements  
  int<lower=1,upper=k> id[n]; // coin ID's  
  real a0;  
  real b0;  
}  
  
parameters {  
  real<lower = 0.0, upper = 1.0> p[k];  
}  
  
model {  
  p ~ beta(a0, b0);  
  for (i in 1:n) {  
    y[i] ~ bernoulli(p[id[i]]);  
  }  
}  
  
generated quantities {  
  real is_best[k];  
  for (i in 1:k) { is_best[i] = 0; }  
  is_best[sort_indices_desc(p)[1]] = 1;  
}
```

Primer (Model Bernoulli-Beta za več skupin)

```
# Prepare dataset for Stan -----
stan_data <- list(y = c(coin1, coin2, coin3, coin4),
                  id = c(rep(1, 100), rep(2, 20), rep(3, 10), rep(4, 4)),
                  n = 134, k = 4, a0 = 0.5, b0 = 0.5)
```

```
# Sample -----
output <- stan(file = "../STANMODELS/bernoulli_beta_multi.stan",
               data = stan_data,
               iter = 5500, warmup = 500,
               chains = 1,
               seed = 0)
```

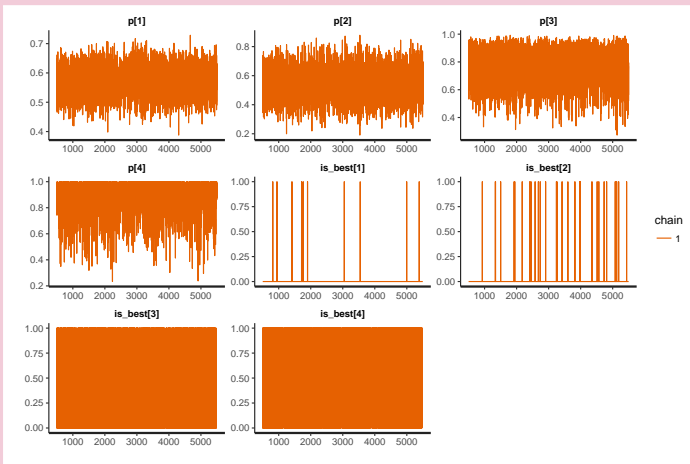
Kako bi vi izbrali?

Naši podatki (uspešni/vsi meti): 57/100, 11/20, 8/10, 4/4

Kateri kovanec ima najbolj verjetno najvišjo pričakovano verjetnost? Kako prepričljivo je najboljši?

Primer (Model Bernoulli-Beta za več skupin)

```
> traceplot(output)
```



Primer (Model Bernoulli-Beta)

```
> print(output)
```

Inference for Stan model: bernoulli_beta_multi.

1 chains, each with iter=5500; warmup=500; thin=1;

post-warmup draws per chain=5000, total post-warmup draws=5000.

	mean	se_mean	sd	2.5%	25%	50%	75%	97.5%	n_eff	Rhat
p[1]	0.57	0.00	0.05	0.47	0.54	0.57	0.60	0.66	4541	1
p[2]	0.55	0.00	0.11	0.34	0.48	0.55	0.62	0.75	4283	1
p[3]	0.77	0.00	0.12	0.50	0.69	0.79	0.87	0.96	5000	1
p[4]	0.90	0.00	0.12	0.57	0.86	0.95	0.99	1.00	5000	1
is_best[1]	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	4246	1
is_best[2]	0.01	0.00	0.08	0.00	0.00	0.00	0.00	0.00	4529	1
is_best[3]	0.18	0.01	0.38	0.00	0.00	0.00	0.00	1.00	4279	1
is_best[4]	0.81	0.01	0.39	0.00	1.00	1.00	1.00	1.00	4377	1
lp__	-93.21	0.03	1.52	-97.00	-93.97	-92.88	-92.08	-91.30	1958	1

Samples were drawn using NUTS(diag_e) at Mon Mar 26 13:46:41 2018.

Primer (Neznan model)

```
data {  
  int n;  
  int ng;  
  real y[n];  
  int group[n];  
  real mu_prior_mu;  
  real mu_prior_sd;  
  real sd_prior_lb;  
  real sd_prior_ub;  
}  
  
parameters {  
  real mu[ng];  
  real<lower=0> sigma[ng];  
  real<lower=1> nu;  
}  
  
model {  
  mu ~ normal(mu_prior_mu, mu_prior_sd);  
  sigma ~ uniform(sd_prior_lb, sd_prior_ub);  
  
  target += exponential_lpdf(nu - 1 | 1.0 / 29.0);  
  for (i in 1:n)  
    y[i] ~ student_t(nu, mu[group[i]], sigma[group[i]]);  
}
```

Povzetek

- **Orodje Stan:**

- Samo še statistika in nič več računanja! No, skoraj...
- Temelji na avtomatskem odvajanju in HMC.
- Kličemo ga lahko iz veliko različnih programskih jezikov.

- **Programski jezik Stan:**

- Stan program = opis modela = navodila za izračun $p(\theta|y)$.
- Pravi programski jezik, a namenjen specifični rabi (t. i. probabilistic programming language).

- **Praktični primeri:**

- Veliko svobode pri modeliranju in pisanju lastnih modelov.
- Stan ponuja veliko funkcionalnosti, ki olajšajo sklepanje in diagnostiko, a
- če želimo, lahko vzamemo samo vzorce in z njimi delamo po svoje.