RSTAN MANUAL

What is STAN?

C++ 언어로 작성된 통계적 추론을 위한 확률적 프로그래밍 언어.

STAN 언어는 로그 확률 밀도 함수를 계산하는 명령적 프로그램으로 (베이지안) 통계 모델을 지정하는데 사용된다.

즉, 사용자는 스탠 언어에서 log density function을 지정하고 다음의 결과들을 얻을 수 있다.

- Full Bayesian statistical inference with MCMC sampling
- Approximate Bayesian inference with variational inference
- penalized maximum likelihood estimation with optimization

Installation

- R 4.0 & Windows 기준

Configuring C++ Toolchain

본격적으로 RStan을 설치하기에 앞서, C++ 코드를 compile할 수 있도록 R 설치를 구성해야 한다.

RTools4 Installation

이를 위해 해야될 일은 RTools 를 설치하는 것이다.

설치하는 방법은 https://cran.r-project.org/bin/windows/Rtools/ 를 참조.

Installation of RStan

우선 안전을 위해, 기존 RStan이 있는지 확인하고 이를 삭제하도록 한다.

```
remove.packages("rstan")
if (file.exists(".RData")) file.remove(".RData")
```

그 다음 R을 재시작한다.

Verify Installation

설치가 제대로 되었는지 확인하기 위해 다음의 example 코드를 실행해본다.

```
example(stan_model, package = "rstan", run.dontrun = TRUE)
```

다음의 Warning Message는 딱히 문제가 되는 것이 아니기 때문에 무시해도 괜찮다.

```
Warning message:
In system(paste(CXX, ARGS), ignore.stdout = TRUE, ignore.stderr = TRUE) :
  'C:/rtools40/usr/mingw_/bin/g++' not found
```

그러나, 다음과 같은 error가 발생한다면, 문제가 있는 것.

```
Error in compileCode(f, code, language = language, verbose = verbose) :
  C:/rtools40/mingw64/bin/../lib/gcc/x86_64-w64-
mingw32/8.3.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:/Program Files/R/R-
4.0.3/library/rstan/lib/x64/libStanServices.a(stan_fit.o):stan_fit.cpp:
(.rdata$_ZZN5boost4math6detail9bessel_j0IeEET_S3_E2P1[_ZZN5boost4math6detail9bes
sel_j0IeEET_S3_E2P1]+0x0): multiple definition of
`boost::math::detail::bessel_j0<long double>(long double)::P1';
file1da012a9cf.o:file1da012a9cf.cpp:
(.data$_ZZN5boost4math6detail9bessel_j0IeEET_S3_E2P1[_ZZN5boost4math6detail9bess
el_j0IeEET_S3_E2P1]+0x0): first defined
hereC:/rtools40/mingw64/bin/../lib/gcc/x86_64-w64-
mingw32/8.3.0/../../../../x86_64-w64-mingw32/bin/ld.exe: C:/Program Files/R/R-
4.0.3/library/rstan/lib/x64/libStanServices.a(stan_fit.o):stan_fit.cpp:
(.rdata$_ZZN5boost4math6detail9bessel_j0IeEET_S3_E2QC[_ZZN5boost4math6detail9bes
sel_j0IeEET_S3_E2QC]+0x0): multiple definition of
`boost::math::detail::bessel_j0<long double>(long double)::QC';
file1da012a9cf.o:file1da012a9cf.cpp:(.data$_ZZN5boos
```

이 경우에는 R을 재시작하고 rstan과 StanHeaders를 다음과 같은 방법으로 재설치한다.

```
# Compile packages using all cores
Sys.setenv(MAKEFLAGS = paste0("-j",parallel::detectCores()))

install.packages(c("StanHeaders","rstan"),type="source")
```

※ RTools가 제대로 설치되지 않은 경우에도 위와 같은 error가 발생할 수 있다.

기본 사용 옵션

Loading the package

패키지 이름은 rstan이다.

```
library("rstan") # observe startup messages
```

Options

- 만약 multicore 컴퓨터를 사용하고 있고, RAM을 모델 추정시 병렬적으로 사용하고 싶다면 다음의 코드를 실행하면 된다.

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

- 다음의 코드를 실행하면 unchanged stan program들이 recompling되는 것을 방지해준다.

  ※ 이 경우 rds 파일이 생성된다.

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

Tutorial

Stan code 기본 구성

Stan 코드의 기본 구성은 data, parameters, model 3개의 블록으로 되어있다.

data 블록에서는 관측 데이터를 선언하고, parameters 블록에는 샘플링하고 싶은 모수를 선언한다. model 블록에는 우도(likelihood)와 사전분포(prior distribution)를 기술한다.

다음은 simple linear regression의 예시.

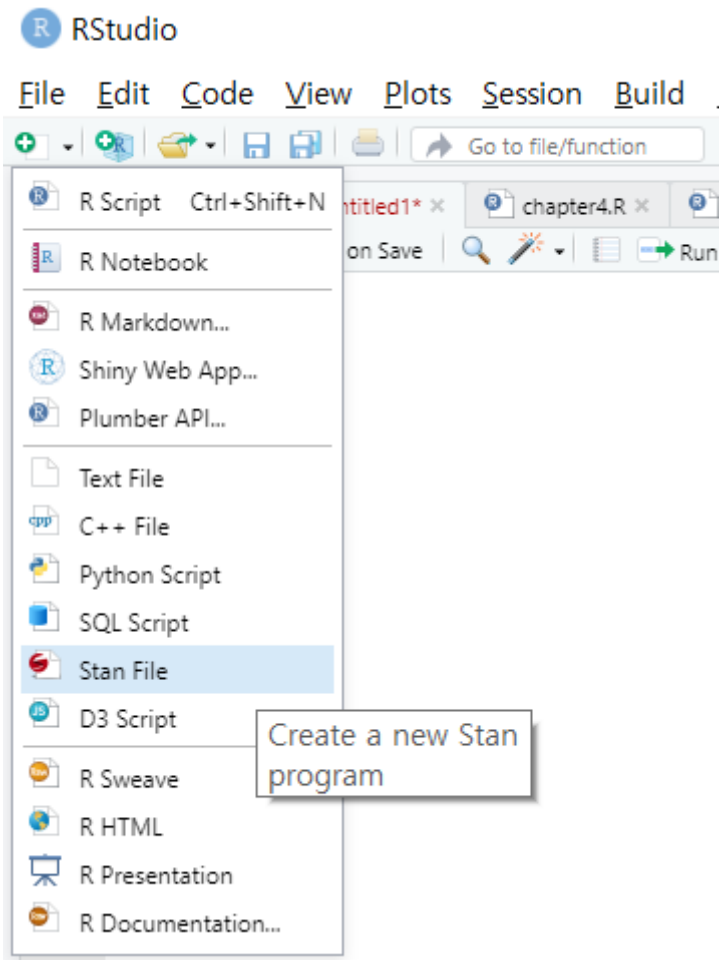
```
data {
  int N;
  real X[N];
  real Y[N];
}
parameters {
  real a;
  real b;
  real<lower = 0> sigma;
}
model {
  for (n in 1:N) {
    Y[n] ~ normal(a + b * X[n], sigma);
  }
}
```

Stan code의 저장

Stan code를 기술하고 저장하는 방법은 세 가지가 있다.

1) 별도의 파일로 저장, 2) R 코드 안에서 정의, 3) R Markdown 문서 안에 포함

1) 별도의 파일로 저장



다음과 같이 stan file을 새로 생성하여 stan code를 작성한 뒤 저장한다.

2) R 코드 안에서 정의

```
lm_stan <- '
data {
  int N;
  real X[N];
  real Y[N];
}
parameters {
  real a;
  real b;
  real<lower = 0> sigma;
}
model {
  for (n in 1:N) {
    Y[n] ~ normal(a + b * X[n], sigma);
  }
}
'
```

다음과 같이 '' 를 이용하여 R 코드 안에서 바로 정의해버릴 수도 있다.

3) R Markdown 문서 안에 포함

```{stan, output.var="lm_stan"}
data {
  int N;
  real X[N];
  real Y[N];
}
parameters {
  real a;
  real b;
  real<lower = 0> sigma;
}
model {
  for (n in 1:N) {
    Y[n] ~ normal(a + b * X[n], sigma);
  }
}
```

Stan Code를 사용한 Model Fitting

stan code를 이용하여 model fitting을 하는 데에 사용되는 대표적인 function은 stan이다.

단, 앞서 언급했던 stan code의 저장 방식에 따라 stan code를 불러오는 방법은 두 가지로 나뉜다.

예시를 통해 살펴보자.

1) 별도의 Stan code file을 저장한 경우

앞서 별도의 stan code file을 만들어서 저장했을 때 file의 이름을 lm_stan.stan 으로 저장했다고 가정하자.

이 때 stan function을 이용하여 model fitting을 하는 식은 다음과 같다.

```
fit <- stan(file = 'lm_stan.stan', data = data_list)
```

※ 이 때 data에 들어가는 data는 우리가 stan code에서 정의했던 data 블록에 들어가는 요소들이 list의 형식으로 모두 들어가야 하며, 이름들도 똑같이 대응되어야 한다.

```
#ex)
data_list = list(N = nrow(cars), X = cars$speed, Y = cars$dist)
```

2) R 코드 내에서 정의하거나 R Markdown 내에서 정의한 경우

이 경우에는 stan code를 별도의 변수에 저장했으므로 이를 이용해야 한다.

앞에서 사용했던 file argument 대신에 model_code argument를 사용하면 된다.

```
fit <- stan(model_code = lm_stan, data = data_list)
```

※ 원하는 경우 stan function에 chain, iteration number, warmup, thinning , seed등 다양한 option들을 설정해줄 수 있다.

기본적으로는 4 chain에 iter=2000, warmup = 1000, thin = 1이 적용된다.

Fitting 후 결과 출력

summary

fitting 결과를 summary를 통해 확인하면 stan code에서 설정한 parameter들의 mean, se_mean, sd, quantile, effective sample size, Rhat 값들을 확인할 수 있다.

chain들의 결과를 통합한 summary와 각 chain별 summary가 모두 제공된다.

만약 통합된 summary 결과를 원한다면, 다음의 code를 이용하면 된다.

```
summary(fit)$summary
```

만약 chain, iteration, warmup, thin 등의 값들을 확인하고 싶다면

단순히

```
fit
```

을 입력하면 된다.

Trace Plot and Posterior Density Plot

- Trace plot을 확인하고 싶다면 traceplot function을 이용하면 된다.

  ```
  traceplot(fit, inc_warmup = TRUE)
  ```

  ※ inc_warmup = TRUE → warmup 을 포함한다.

  만약 어떤 parameter의 traceplot을 따로 확인하고 싶다면, pars argument를 사용하면 된다.

  ```
  traceplot(fit, pars = 'sigma', inc_warmup = TRUE)
  ```

- Posterior Density Plot을 확인하고 싶다면 plot function을 사용하면 된다. 이 때 show_density argument를 TRUE하지 않는다면, 해당 parameter의 80% ci_level 범위만 나타남에 유의.

  ```
  plot(fit)
  ```

마찬가지로 만약 특정 parameter의 plot만 따로 확인하고 싶다면, pars를 사용하면 된다.

```
plot(fit, pars = 'sigma', show_density = TRUE)
```