Quant III

Lab 7: Beyesian: Computational Tools

Junlong Aaron Zhou

October 29, 2020

Outline

- Midterm
- RStan

Midterm

- Good.
- Common problems:
  - 1e, what is an informative prior?
  - 2, what is MLE?
  - 5c,5d. What's the problem in likelihood?

Intro to Stan Language

- Different software for Bayesian modeling; Stan is one of the best
- Probabilistic language: random variables are basic elements
- Stan defines statistical models as conditional probability distributions $p(\theta|y;x)$

Stan

- Stan programs include variable type declarations and statements
- Observed r.v. declared as data; unobserved - as parameters
- Workflow:
  1. Stan code translated to a C++ program
  2. compilation to an executable
  3. execution
- Statements order matters!
- More about Rstan: **here**.

Stan: Order

- (OPTIONAL:) functions
- data
- (OPTIONAL:) transformed data
- parameters
- (OPTIONAL:) transformed parameters
- model
- (OPTIONAL:) generated quantities

Stan Example: What's going on here?

```
data{
  int N ;
  vector[N] y ;
}
parameters{
  real mu ;
}
model{
  mu ~ normal(0, 1) ; // Normal with mean 0,
                      // standard deviation 1.
  y ~ normal(mu, 1) ;
}
```

Stan variable types (1)

- 2 primitive types
  - real
  - int
- Multiple unconstrained "container" types
  - vector (always column vector)
  - matrix
  - row vector
  - array

Stan variable types (1)

- 2 primitive types
  - real
  - int
- Multiple unconstrained “container” types
  - vector (always column vector)
  - matrix
  - row vector
  - array
- Multiple constrained “container” types
  - simplex
  - ordered
  - corr matrix
  - etc.

Stan variable types (2)

- Vectors, matrices, arrays are not assignable to each other
- Vectors and matrices contain only real numbers Arrays are the most flexible (can be arrays of int, real, matrices...)

Stan variable types (2)

- Vectors, matrices, arrays are not assignable to each other
- Vectors and matrices contain only real numbers Arrays are the most flexible (can be arrays of int, real, matrices. . .)
- Why using vector and matrix types?
- But matrix and vector types only are allowed:
  - with matrix arithmetic operations
  - with linear algebra functions
  - as parameters of multivariate functions

Stan variable types (3)

```
int N ;
```

Stan variable types (3)

```
int N ;

real y ;
```

Stan variable types (3)

```
int N ;

real y ;

vector[10] mu ;
```

Stan variable types (3)

```
int N ;

real y ;

vector[10] mu ;

matrix[2, 3] X ;
```

Stan variable types (3)

```
int N ;

real y ;

vector[10] mu ;

matrix[2, 3] X ;

real theta[5] ;
int y[5] ;
matrix[2,3] X[N] ;// (array of length N, each element is a 2x3
```

Stan variable types (4)

- Referencing elements of containers

```
matrix[2,3] X ; X[2] ; X[2,3] ; X[2][3]
matrix[2,3] X[N] ; X[2][1,1]
```

Stan variable types (5): Array here?

```
data{
  int N ;
  vector[N] y ;
}
parameters{
  real mu ;
}
model{
  mu ~ normal(0, 1) ;
  y ~ normal(mu, 1) ;
}
```

Stan variable types (6):What's wrong?

```
data{
  int N ;
  vector[N] y ;
}
parameters{
  real lambda ;
}
model {
  lambda ~ normal(0, 1) ;
  y ~ poisson(lambda) ;
}
```

Stan variable types (7)

Constraining with $<>$

```
int<lower=0, upper=1> y[5] ;
real<lower=0> theta ;
```

Stan variable types (8):Poisson example fixed

```
data{
  int N ;
  vector[N] y ;
}
parameters{
  real<lower=0> lambda ;
}
model {
  lambda ~ normal(0, 1) ;
  y ~ poisson(lambda) ;
}
```

Target syntax (1)

- Stan evaluates a log probability function of a set of parameters
- Often, without additive constants
- y $\sim$ poisson(lambda) doesn't do any sampling!

Target syntax (1)

- Stan evaluates a log probability function of a set of parameters
- Often, without additive constants
- y $\sim$ poisson(lambda) doesn't do any sampling!
- `target += poisson_lpdf(y | lambda)`
- `target +=` syntax tells Stan NOT to ignore constants (slows down computations a little)

Target syntax (2): What's the output here?

```
parameters{
  real y ;
}
model{
  target += -0.5 * square(y) ;
}
```

Target syntax (3): What's the output here?

```
parameters {
  real y ;
}
model {
   for (n in 1:10) {
    target += -0.5 * square(y) ;
   }
}
```

Plan for later today

- Write a string that represents a Stan program
- Use **rstan::stan()**: Compile and execute the code, sample from the posterior
- Assess model convergence graphically and numerically
- Get posterior distributions of parameters