Woods Hole

Specifying model parameters

Anders Nielsen

Model parameters are what we want to estimate

- We setup a list of model parameters from within R
- The initial value is specified from R
- All the cpp code does is to evaluate the function (and derivatives)
- Optimization is done from R, so values need to be passed from and to many times.
- Simplest example:

```
library(TMB)
compile("scalar.cpp")
dyn.load(dynlib("scalar"))

data <- list()

param <- list()
param$mu <- 0

obj <- MakeADFun(data, param, DLL="scalar")
opt <- nlminb(obj$par, obj$fn, obj$gr)
opt$par
```

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator()()
{
  PARAMETER(mu);
  Type nll = pow(Type(42)-mu,2);
  return nll;
}
```

Simple bounds on a parameter (from R)

- Consider the model:

$$X \sim \text{Bin}(100, p)$$

- Let's say we have observed $X = 2$
- Want to estimate our model parameter p

```r
library(TMB)
compile("p1.cpp")
dyn.load(dynlib("p1"))
data <- list()
data$X <- 2
param <- list()
param$p <- .5

obj <- MakeADFun(data, param, DLL="p1",
                 silent=TRUE)
opt <- nlminb(obj$par, obj$fn, obj$gr,
              lower=c(0), upper=c(1))
summary(sdreport(obj))
```

```cpp
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator()()
{
  DATA_SCALAR(X);
  PARAMETER(p);
  Type nll = -dbinom(X,Type(100),p,true);
  return nll;
}
```

- We get:

```
Estimate Std. Error
p 0.02   0.01398284
```

- Now we want to make a 95% confidence interval — see the problem?

Simple bounds on a parameter (from cpp)

- Consider same model and observation, but now parametrized as:

$$X \sim \text{Bin}(100, p), \quad \text{where } \text{logit}(p) = \alpha$$

- Now we write as:

```
library(TMB)
compile("p2.cpp")
dyn.load(dynlib("p2"))
data <- list()
data$X <- 2
param <- list()
param$alpha <- 0

obj <- MakeADFun(data, param, DLL="p2",
                 silent=TRUE)
opt <- nlminb(obj$par, obj$fn, obj$gr)
summary(sdreport(obj))
```

```
#include <TMB.hpp>

template<class Type>
Type trans(Type x){
  return exp(x)/(Type(1)+exp(x));
}

template<class Type>
Type objective_function<Type>::operator()()
{
  DATA_SCALAR(X);
  PARAMETER(alpha);
  Type p=trans(alpha);
  Type nll = -dbinom(X,Type(100),p,true);
  ADREPORT(p)
  return nll;
}
```

```
       Estimate Std. Error
alpha -3.89182  0.7142857
```

- Now we can make a 95% confidence interval:

```
> x <- -3.89182+0.7142857%o%c(-2,2)
> exp(x)/(1+exp(x))
            [,1]       [,2]
[1,] 0.004867035 0.07847509
```

Exercise

Exercise 1: Suggest how to use transformation to parametrize a parameter that is

a) only positive

b) only negative

c) between 2 and 5

d) an increasing vector

Solution: Consider the following transformations

a) $\theta = e^{\alpha}, \quad \text{where } \alpha \in \mathcal{R}$

b) $\theta = -e^{\alpha}, \quad \text{where } \alpha \in \mathcal{R}$

c) $\theta = 3e^{\alpha}/(1+e^{\alpha})+2, \quad \text{where } \alpha \in \mathcal{R}$

d) $\theta = (e^{\alpha_1}, e^{\alpha_1}+e^{\alpha_2}, \ldots, e^{\alpha_1}+\cdots+e^{\alpha_n}), \quad \text{where } \alpha \in \mathcal{R}^n$

Often we have more than one

- The following parameter types are available:

Template Syntax	**C++ type**	**R type**
`PARAMETER(name)`	`Type`	`numeric(1)`
`PARAMETER_VECTOR(name)`	`vector<Type>`	`vector`
`PARAMETER_MATRIX(name)`	`matrix<Type>`	`matrix`
`PARAMETER_ARRAY(name)`	`array<Type>`	`array`

- Just like with data we can specify a list of possibly many parameter objects
- Naturally we need to match each parameter object on the cpp side

Exercise: Probability vector

- For a single probability parameter we can use the inverse logit transformation

$$p = \exp(\alpha)/(1+\exp(\alpha)), \quad \text{where } \alpha \in \mathcal{R}$$

- For a probability vector $p = (p_1, \ldots, p_n) \in]0,1[^n$ with $\sum p = 1$ we can use the following transformation:

$$p = \begin{pmatrix} \exp(\alpha_1)/(1+\sum_{i=1}^{n-1}\exp(\alpha_i)) \\ \exp(\alpha_2)/(1+\sum_{i=1}^{n-1}\exp(\alpha_i)) \\ \vdots \\ \exp(\alpha_{n-1})/(1+\sum_{i=1}^{n-1}\exp(\alpha_i)) \\ 1-\sum_{i=1}^{n-1}\exp(\alpha_i)/(1+\sum_{i=1}^{n-1}\exp(\alpha_i)) \end{pmatrix} \quad \text{where } \alpha \in \mathcal{R}^{n-1}$$

- Assume we observed the vector (128,158,92,122) from a four dimensional multinomial.

- write the code to estimate the p-vector

Solution

```r
library(TMB)
compile("p3.cpp")
dyn.load(dynlib("p3"))
data <- list()
data$X <- c(128,158,92,122)

param <- list()
param$alpha <- c(0,0,0)

obj <- MakeADFun(data, param, DLL="p3",
                 silent=TRUE)
opt <- nlminb(obj$par, obj$fn, obj$gr)
summary(sdreport(obj))
```

```cpp
#include <TMB.hpp>

template<class Type>
vector<Type> trans(vector<Type> alpha){
  int dim=alpha.size();
  vector<Type> p(dim+1);
  vector<Type> expa=exp(alpha);
  Type s=sum(expa);
  Type lastp=1;
  for(int i=0; i<dim; ++i){
    p(i)=expa(i)/(Type(1)+s);
    lastp-=p(i);
  }
  p(dim)=lastp;
  return p;
}

template<class Type>
Type objective_function<Type>::operator()()
{
  DATA_VECTOR(X);
  PARAMETER_VECTOR(alpha);
  vector<Type> p=trans(alpha);
  Type nll = -dmultinom(X,p,true);
  ADREPORT(p);
  return nll;
}
```

Collapsing parameters, or fixing them

- The `map` argument of the `MakeADFun` can be used to couple elements in a parameter object
- If we have a parameter vector `alpha` of length 4, then the statement:

```
obj <- MakeADFun(data, param, map=list(alpha=factor(c(1,2,3,3))))
```

- will collapse the last two parameters.
- They will be initialized to the mean of the last two initializations
- The optimizer will estimate a common value for both parameters
- This structure is perfect for testing many model hypotheses
- In addition if `NA` is set, as in:

```
obj <- MakeADFun(data, param, map=list(alpha=factor(c(1,2,NA,4))))
```

- then the optimizer treat that parameter (here the third) as fixed.

Exercise: Use of the map argument

- Consider the data set `InsectSprays`, which is available in R
- We will use the model:

$$\text{count}_i \sim \text{Pois}(\lambda_i) \ , \quad \text{where } \log \lambda_i = \alpha(\text{spray}_i)$$

- This can be implemented as:

```
library(TMB)
compile("insect.cpp")
dyn.load(dynlib("insect"))

# for data we use the built-in InsectSprays
param <- list(
  logAlpha=rep(0,nlevels(InsectSprays$spray))
)
obj <- MakeADFun(InsectSprays, param, DLL="insect")
opt <- nlminb(obj$par, obj$fn, obj$gr)
```

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator()()
{
  DATA_VECTOR(count);
  DATA_FACTOR(spray);
  PARAMETER_VECTOR(logAlpha);
  Type nll = 0;
  for(int i=0; i<count.size(); ++i){
    Type lambda = exp(logAlpha(spray(i)));
    nll += -dpois(count(i),lambda,true);
  }
  return nll;
}
```

- Use the `map` argument to test the hypothesis that spray $\alpha(A) = \alpha(B) = \alpha(F)$
- Can the mean count for the spray ‘A’, ‘B’ and ‘F’ and be assumed to be equal to 15.

(try to test these hypothesis without modifying the cpp file)

Solution

```
library(TMB)
compile("insect.cpp")
dyn.load(dynlib("insect"))

# for data we use the built-in InsectSprays
param <- list(
  logAlpha=rep(0,nlevels(InsectSprays$spray))
)
obj <- MakeADFun(InsectSprays, param, silent=TRUE)
opt <- nlminb(obj$par, obj$fn, obj$gr)

map1=list(logAlpha=factor(c(1,1,2,3,4,1)))
obj1 <- MakeADFun(InsectSprays, param, map=map1, silent=TRUE)
opt1 <- nlminb(obj1$par, obj1$fn, obj1$gr)
1-pchisq(2*(opt1$obj-opt$obj),2)
#
#  0.3982677
#
map2=list(logAlpha=factor(c(NA,NA,2,3,4,NA)))
param2<-param
param2$logAlpha[c(1,2,6)]<-log(15)
obj2 <- MakeADFun(InsectSprays, param2, map=map2, silent=TRUE)
opt2 <- nlminb(obj2$par, obj2$fn, obj2$gr)
1-pchisq(2*(opt2$obj-opt1$obj),1)
#
# 0.4410911
#
```