Simulating and estimating the hierarchical lengths example from Bence

Christopher L. Cahill

Quantitative Fisheries Center

R code for simulating a mixed effects model: `lengths.R`

```r
n_pond = 8
n_fish = 10
mu = 150 # assume average size among ponds is 150 mm
sd_among = 50 # among lake (shared) sd
sd_within = 10 # within lake sd
pond = rep(1:n_pond, each = n_fish)
print(pond)
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4
4
[39] 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8
8
[77] 8 8 8 8
```

`lengths.R` cont'd

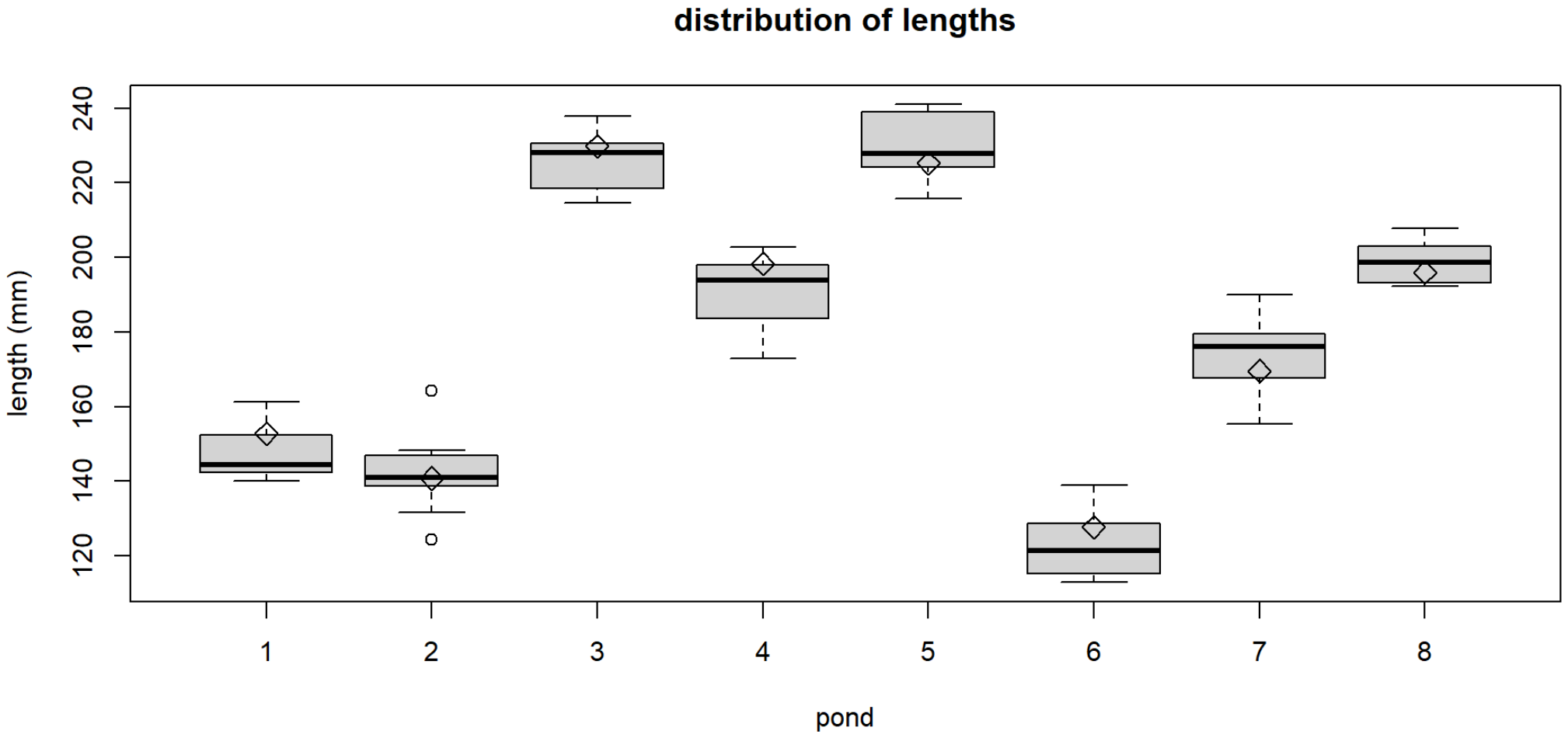
```
# step 1, calculate pond mean lengths from hyperprior:
set.seed(31)
mu_pond = rnorm(n_pond, mu, sd_among) # average length in each lake
```

lengths.R cont'd

```
# step 2, calculate fish lengths | on mu_pond:
y_obs = rep(NA, n_pond * n_fish)

# loop through data, generate a random fish length | mu_pond
for (i in 1:length(y_obs)) {
  y_obs[i] = rnorm(1, mu_pond[pond[i]], sd_within)
}
```

Look at the simulated data



TMB code for estimating the lengths model: `lengths.cpp`

```cpp
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y_obs);       // observed data
  DATA_INTEGER(n_pond);
  DATA_IVECTOR(pond);       // integer vector indicating pond

  PARAMETER(ln_mu);         // average length among lakes
  PARAMETER(ln_sd_among);   // sd among ponds
  PARAMETER(ln_sd_within);  // sd within ponds
  PARAMETER_VECTOR(eps);    // random effects - deviations from mu

  int n_obs = y_obs.size();
  Type mu = exp(ln_mu);
  Type sd_among = exp(ln_sd_among);
  Type sd_within = exp(ln_sd_within);

  Type jnll = 0.0;

  // Pr(random coefficients)
  for(int i = 0; i < n_pond; i++){
    jnll -= dnorm(eps(i), Type(0.0), sd_among, true);
  }
  // more code below
```

TMB code: `lengths.cpp`

```
  // continued from previous slide

  // Pr(data conditional on fixed and random effect values)
  for(int i = 0; i < n_obs; i++){
    jnll -= dnorm(y_obs(i), mu + eps(pond(i)), sd_within, true);
  }

  return jnll;
}
```

R code: `lengths.R`

```r
library(TMB)
compile("lengths.cpp")
```

```
[1] 0
```

```r
dyn.load(dynlib("lengths"))

data = list(
  y_obs = y_obs,
  n_pond = n_pond,
  pond = pond - 1 # look here, this is important
)

parameters = list(
  ln_mu = log(150),
  ln_sd_among = log(50),
  ln_sd_within = log(50),
  eps = rep(0, n_pond)
)

obj = MakeADFun(data, parameters, random = "eps", DLL = "lengths")
```

Testing objective function

```
1 obj$fn() # return the objective f(x) value
```

```
Optimizing tape... Done
iter: 1  value: 429.3855 mgc: 0.3160354 ustep: 1
iter: 2  mgc: 5.182486e-17
```

```
[1] 400.3294
attr(,"logarithm")
[1] TRUE
```

Testing gradients

```
1 obj$gr() # examine par gradients
```

```
iter: 1  mgc: 5.182486e-17
Matching hessian patterns... Done
outer mgc:  69.99745
```

```
[1] -12.631046   1.480762  69.997450
```

Run the optimization:

```
1 opt = nlminb(obj$par, obj$fn, obj$gr)
```

```
iter: 1  mgc: 5.182486e-17
iter: 1  mgc: 5.182486e-17
outer mgc:  69.99745
iter: 1  value: 356.1635 mgc: 0.8939608 ustep: 1
iter: 2  mgc: 6.782769e-16
iter: 1  value: 2960.189 mgc: 1329.285 ustep: 1
iter: 2  mgc: 5.255102e-13
iter: 1  mgc: 6.782769e-16
outer mgc:  56.6637
iter: 1  value: 605.3189 mgc: 2.211058 ustep: 1
iter: 2  mgc: 2.735832e-14
iter: 1  value: 338.4739 mgc: 0.02171076 ustep: 1
iter: 2  mgc: 1.028691e-15
iter: 1  mgc: 1.028691e-15
outer mgc:  44.02805
```

Get standard deviations

```
1 sdr = sdreport(obj)
```

```
iter: 1  mgc: 2.669739e-15
outer mgc:  2.076325e-05
iter: 1  value: 322.2081 mgc: 0.02397546 ustep: 1
iter: 2  mgc: 2.359224e-15
outer mgc:  0.1897302
iter: 1  value: 322.2081 mgc: 0.0239515 ustep: 1
iter: 2  mgc: 1.852685e-15
outer mgc:  0.1891672
iter: 1  value: 322.208 mgc: 8.261608e-05 ustep: 1
iter: 2  mgc: 1.132774e-15
outer mgc:  0.0158112
iter: 1  value: 322.2079 mgc: 8.278147e-05 ustep: 1
iter: 2  mgc: 1.619798e-15
outer mgc:  0.01583635
iter: 1  value: 322.216 mgc: 8.261608e-05 ustep: 1
```

Get standard deviations

```
1 print(sdr)
```

```
sdreport(.) result
             Estimate Std. Error
ln_mu        5.187085 0.07265310
ln_sd_among  3.601978 0.25138866
ln_sd_within 2.156570 0.08333332
Maximum gradient component: 2.076325e-05
```

Check diagnostics

```
# check gradients and make sure pdHess
opt$convergence # 0 is good
```

```
[1] 0
```

```
final_gradient <- obj$gr(opt$par)
```

```
iter: 1  mgc: 2.669739e-15
outer mgc:  2.076325e-05
```

```
if (any(abs(final_gradient) > 0.001) || sdr$pdHess == FALSE) {
  message("Model did not converge: check results")
} else {
  message("Model diagnostics consistent with convergence")
}
```

Plot fits vs. data