Fitting models via maximum likelihood using RTMB

Chris Cahill and Jim Bence

21 July 2023

Quantitative Fisheries Center
MICHIGAN STATE UNIVERSITY

Outline

- Demonstrate a mind-blowing advance with AD
- Show a few examples that may be useful
  - Start with equations, move to code
- Talk about debugging via `browser()`
- Tips and trickery
- Code for all of this available at: https://github.com/QFCatMSU/RTMB/tree/main

What is RTMB?

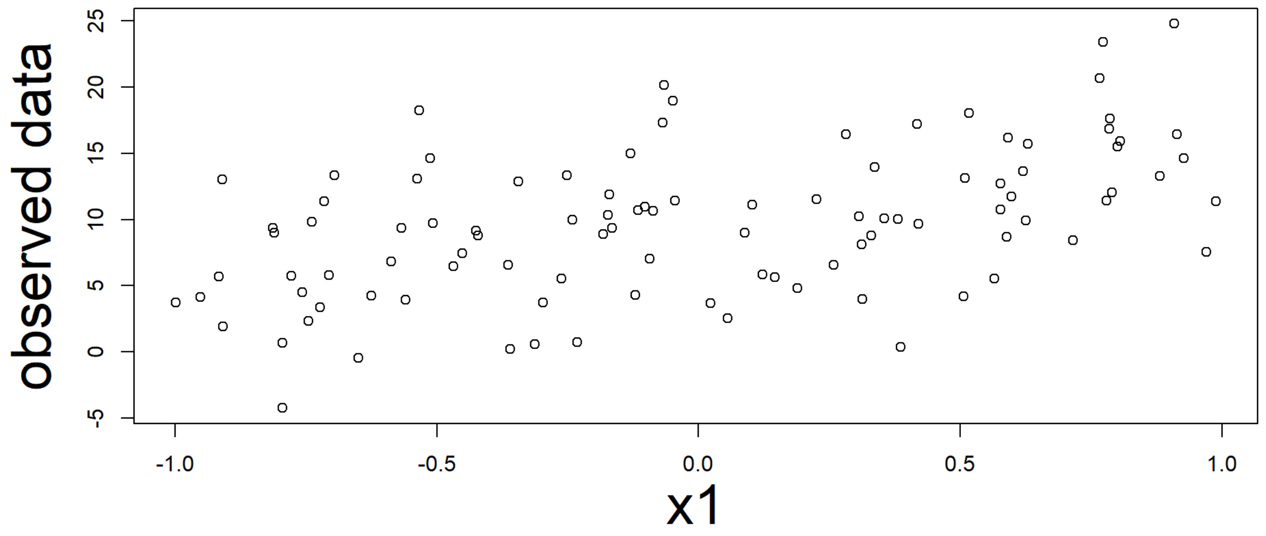
- RTMB is a new package that provides a native R interface for a subset of TMB so you can avoid coding in C++
- See https://kaskr.r-universe.dev/RTMB
- In all applications we have tried all TMB functionality was available!
- Because code is all in R, both easier to code and for others to read that code - a game changer!
- A game changer *if* you know how to code in R, create an objective f(x) for your model
- No compiling or compiling errors!
- Bottom line: less time developing and testing models, more intuitive code

Linear regression in RTMB

- The math:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \text{where} \quad \epsilon_i \sim \mathrm{N}(0, \sigma)$$

```r
par(mar=c(5,6,4,1) + 0.1)
plot(y_obs~x1, xlab = "x1", ylab = "observed data", cex.lab = 2.5)
```



Linear regression (RTMB)

```r
# set up tagged data + parameters lists:
data = list(
  n = n,
  y_obs = y_obs,
  x1 = x1
)

pars = list(
  b0 = 1,
  b1 = 1,
  log_sd = log(3)
)
```

- see `linreg.r`

Linear regression (RTMB)

```r
library(RTMB)

# write an objective function returning negative log-likelihood
f = function(pars) {
  getAll(data, pars) # replaces DATA_XX, PARAMETER_YY
  y_pred = b0 + b1 * x1
  nll = -sum(dnorm(y_obs, y_pred, exp(log_sd), log = TRUE))
  nll
}

obj = MakeADFun(f, pars)
obj$fn() # objective function
```

```
[1] 777.5154
```

```r
obj$gr() # gradients
```

```
outer mgc:  1051.521

          [,1]     [,2]      [,3]
[1,] -96.88602 -13.2133 -1051.521
```

- Stick to base R

Linear regression (RTMB)

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

```
outer mgc:  1051.521
outer mgc:  54.72716
outer mgc:  36.6083
outer mgc:  21.88026
outer mgc:  35.7811
outer mgc:  5.686432
outer mgc:  1.353547
outer mgc:  1.508322
outer mgc:  0.08695211
outer mgc:  0.04554657
outer mgc:  0.0305969
outer mgc:  0.002009656
outer mgc:  1.470605e-05
```

Linear regression (RTMB)

```r
sdr = sdreport(obj)
```

```
outer mgc:  1.470605e-05
outer mgc:  0.004344401
outer mgc:  0.004344313
outer mgc:  0.001393873
outer mgc:  0.00140084
outer mgc:  0.1997855
outer mgc:  0.2002149
```

- We are done.

Access fit

```r
opt
```

```
$par
      b0       b1   log_sd 
9.730622 4.775419 1.568146 

$objective
[1] 298.7085

$convergence
[1] 0

$iterations
[1] 12

$evaluations
function gradient 
```

Access fit

```r
sdr
```

```
sdreport(.) result
       Estimate Std. Error
b0     9.730622 0.47978081
b1     4.775419 0.84596427
log_sd 1.568146 0.07071065
Maximum gradient component: 1.470605e-05
```

RTMB scales to (much) more complicated models

von Bertalanffy growth model: the math

$$l_i = l_\infty \left(1 - e^{-k(a_i - t_0)}\right) + \varepsilon_i$$

$$\varepsilon_i \sim \mathrm{N}\left(0, \sigma^2\right)$$

RTMB objective f(x) for a von Bertalanffy growth model:

```r
f = function(pars) {
  getAll(data, pars)
  linf = exp(log_linf)
  vbk = exp(log_vbk)
  sd = exp(log_sd)
  l_pred = linf * (1 - exp(-vbk * (age_i - t0)))
  nll = -sum(dnorm(l_obs_i, l_pred, sd, TRUE))
  REPORT(linf)
  REPORT(vbk)
  ADREPORT(sd)
  nll
}
```

- see `vonB.r`
- can use REPORT(), ADREPORT()

Poisson GLMM: the math

$$
\begin{aligned}
y_{i,site} &\sim \text{Poisson}\left(\lambda_{i,site}\right) \\
\log(\lambda_{i,site}) &= \beta_0 + \beta_1 \cdot x_1 + \epsilon_{site} \\
\epsilon_{site} &\sim \text{N}(0, \sigma^2_{site})
\end{aligned}
$$

- β_0 is global intercept shared among sites
- x_1 is a covariate
- ϵ_{site} is normally distributed random effect

Objective f(x) for a Poisson GLMM:

```r
f = function(pars) {
  getAll(data, pars)
  sd_site = exp(log_sd_site)                                # back transform
  jnll = 0                                                  # initialize
  jnll = jnll - sum(dnorm(eps_site, 0, sd_site, TRUE)) # Pr(random effects)
  lam_i = exp(                                              # link f(x)
    Xmat %*% bvec +                                         # fixed effects
    eps_site                                                # random effects
  )
  jnll = jnll - sum(dpois(yobs, lam_i, TRUE))               # Pr(observations)
  jnll
}
```

- see `glmm.r`

Objective f(x) for a hierarchical normal selectivity model, adapted from Millar and Freyer 1999:

```r
f = function(pars) {
  getAll(data, pars)
  jnll = 0
  jnll = jnll - sum(dnorm(k1_dev, 0, exp(ln_sd_k1), TRUE))
  jnll = jnll - sum(dnorm(k2_dev, 0, exp(ln_sd_k2), TRUE))
  k1 = exp(ln_k1 + k1_dev)
  k2 = exp(ln_k2 + k2_dev)
  sel_mat = phi_mat = matrix(0, nrow(catches), ncol(catches))
  for (i in 1:nrow(sel_mat)) {
    for (j in 1:ncol(sel_mat)) {
      sel_mat[i, j] = exp(-(lens[i] - k1[site[i]] * rel_size[j])^2 /
        (2 * k2[site[i]]^2 * rel_size[j]^2))
    }
  }
  sel_sums = rowSums(sel_mat)
  for (i in 1:nrow(phi_mat)) {
    for (j in 1:ncol(phi_mat)) {
      phi_mat[i, j] = sel_mat[i, j] / sel_sums[i]
    }
  }
  jnll = jnll - sum(catches * log(phi_mat))
  jnll
}
```

Spatially explicit Poisson GLMM:

- see `glmm_spde.r`

Spatial-temporal Poisson GLMM:

- see `glmm_ar1spde.r`

Conventional vonB with random L_∞: math

$$L_{i,j} = L_{\infty_i}\left(1 - \exp(-K\left(a_{i,j} - t_0\right))\right) + \varepsilon_{i,j}$$

$$\varepsilon_{i,j} \sim N\left(0, \sigma^2\right)$$

$$\log(L_{\infty_i}) \sim N\left(\log(L_\infty), \sigma^2_{L\infty}\right)$$

- Note L_∞ is the median and not mean asymptote among ponds
- Also note, the model code uses likelihood equation with equivalent $L_{i,j} \sim N\left(\hat{L}_{i,j}, \sigma^2\right)$

Objective f(x) for a Bence's vonB:

```r
f = function(pars) {
  getAll(data, pars)
  Linfmn = exp(logLinfmn)
  logLinfsd = exp(loglogLinfsd)
  Linfs = exp(logLinfs)
  K = exp(logK)
  Sig = exp(logSig)
  nponds = length(Linfs)
  nages = length(A)
  predL = matrix(0, nrow = nages, ncol = nponds)
  # fill one column (pond) at a time:
  for (i in 1:nponds) {
    predL[, i] = Linfs[i] * (1 - exp(-K * (A - t0)))
  }
  nll = -sum(dnorm(x = L, mean = predL, sd = Sig, log = TRUE))
  nprand = -sum(dnorm(x = logLinfs, mean = logLinfmn, sd = logLinfsd, log = TRUE))
  jnll = nll + nprand
  jnll
}
```

- see `multilinf.r`

Objective f(x) for a Ricker stock-recruit model with a random walk α: math

TODO

Objective f(x) for a Ricker stock-recruit model with a random walk α

```r
f = function(pars) {
  getAll(data, pars)
  jnll =  0 # initialize
  # initial state
  jnll =  jnll - dnorm(alphas[1], exp(log_alpha0), exp(log_sd_alpha), TRUE)
  # walk through the remaining years
  for (t in 2:length(data$year)) {
    jnll =  jnll - dnorm(alphas[t], alphas[t - 1], exp(log_sd_alpha), TRUE)
  }
  # calculate systematic component:
  log_RS_pred =  alphas - beta * S
  # likelihood for observations vs. predictions:
  jnll =  jnll - sum(dnorm(log_RS_obs, log_RS_pred, exp(log_sdr), TRUE))
  REPORT(log_RS_pred)
  ADREPORT(alphas)
  jnll
}
```

- see `rw_ricker.r`

Objective f(x) for a Ricker stock-recruit model with an AR-1 α: math

TODO

Objective f(x) for an AR-1 Ricker α

```r
to_cor =  function(x) { # -inf to inf --> -1 to 1 transform
  2 / (1 + exp(-2 * x)) - 1
}

f =  function(pars) {
  getAll(data, pars)
  n_year =  length(R)
  rho =  to_cor(trans_rho) # back transform
  sd_eps =  exp(log_sd_eps)
  sd_obs =  exp(log_sd_obs)
  jnll =  0
  jnll =  jnll - dnorm(eps_a[1], 0, sqrt(1 - rho^2) * sd_eps, TRUE) # initialize
  for (t in 2:n_year) {
    jnll =  jnll - dnorm(eps_a[t],                # current eps
                         rho * eps_a[t - 1],      # is a function of eps[t-1]
                         sqrt(1 - rho^2) * sd_eps, # + some stationary noise
                         TRUE
                        )
  }
  pred_log_R =  log_alpha + eps_a - beta * S + log(S)
  jnll =  jnll - sum(dnorm(log(R), pred_log_R, sd_obs, TRUE))
  jnll
}
```

- see `ar1_ricker.r`

Debugging

- Because RTMB is written in R, can use debugging tools (!)
- `browser()` allows you to step through and check calculations
  - no longer need `cout` or `REPORT()` calls to check calculations
- Can also use breakpoints
- Jump to demonstration (in-person)

Tips

- Talk about `advector` error messages
    - Often(?) means you are using something that isn't supported by RTMB
- Make sure you are using base R / RTMB f(x)'s
- Hash out each line to find the offending line(s)
- Discuss other oddities

Questions?

- cahill11@msu.edu