Fitting nonlinear models using automatic differentiation in R via RTMB

Christopher Cahill
21 July 2023

Quantitative Fisheries Center
MICHIGAN STATE UNIVERSITY

Outline

- Demonstrate an exciting 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
- All code available at https://github.com/QFCatMSU/RTMB/tree/main

What is RTMB?

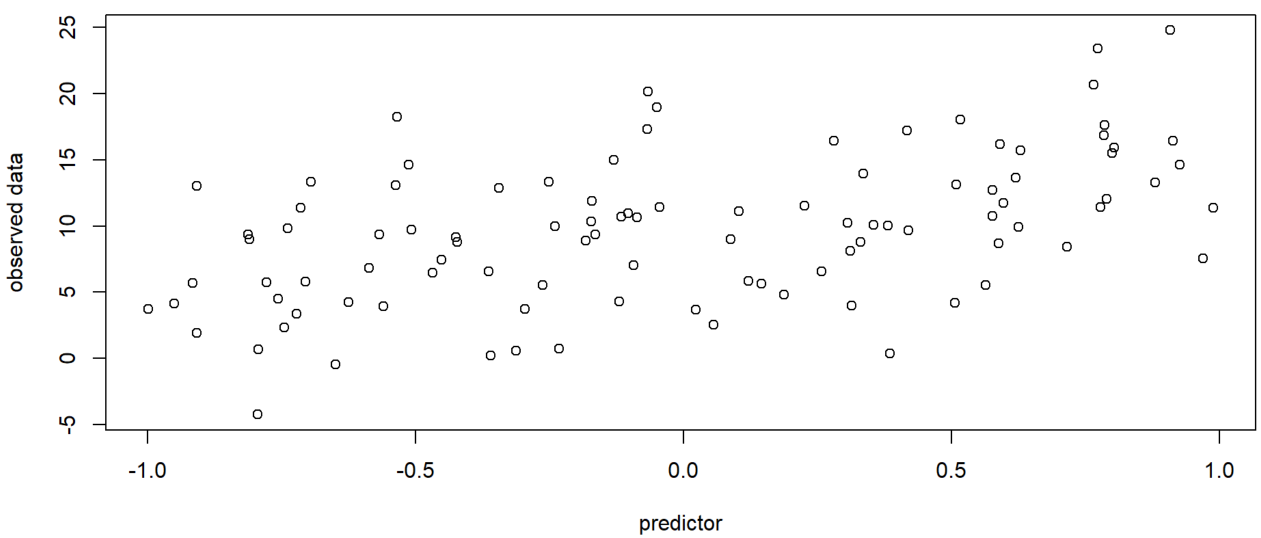
- RTMB is a new package that provides a native R interface for a subset of TMB so you can avoid coding in C++.
- See link: https://kaskr.r-universe.dev/RTMB
- No longer need to code a `.cpp` file to use AD
- All(?) the functionality of TMB
- Easier for others to read the code (no more `.cpp` files)
- A game changer *if* you know how to code in R, create an objective f(x) for your model
- Bottom line: less time developing and testing models, more intuitive code

Linear regression

- The math:

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

```
plot(y_obs~x1, xlab = "predictor", ylab = "observed data")
```



Linear regression (RTMB)

```
# 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)

```
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

```
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 
      15       13 
```

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 works for more complicated models

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()

Objective f(x) for a Poisson hierarchical model:

```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(                                             # 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 selectivity model, see 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
}
```

- see `by_net.r`

Objective f(x) for a spatially explicit GLMM:

```
f = function(pars){
  getAll(data, pars)
  SIGMA = gp_sigma * exp(-dist_sites / gp_theta)
  jnll = 0
  jnll = jnll - sum(dmvnorm(eps_s, SIGMA, TRUE))
  y_hat = exp(beta0 + eps_s) # index on site_i in more complex examples
  jnll = jnll - sum(dpois(y_obs, y_hat), TRUE)
  jnll
}
```

- wut dark majicks is this??!!
- see `grf.r`

Objective f(x) for a Bence's fancy 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`