Introduction to RTMB

Olav Nikolai Risdal Breivik

Thanks to Anders Nielsen for borrowing course material

Norsk Regnesentral
ANVENDT DATAFORSKNING

What is RTMB

- General tool for conducting inference
- Users write the statistical model in plain R
- RTMB sets up machinery for efficiently:
  - Calculating the function value
  - Marginalizing over latent effects efficiently
  - Differentiate the (marginal) likelihood with automatic differentiation
- Then use `nlminb` for a gradient-based search for optimal parameters
- Many neat features for simulation and validation

Motivation

- Standard models are useful but should not limit us
  - Formula interfaces are sometimes frustrating
  - It is often a giant task to move beyond
- RTMB makes it easy to implement non-standard models
  - You just need to write the likelihood in R
  - Gradients and Laplace approximations are calculated by RTMB.
- Inference goes very fast
- You are in full control
  - You do not need others to extend their package
- Simple to validate your model

Non-standard models

- Models where you need to write your own likelihood
- Models you cannot write in one line in R
- Non-trivial non-linearities
- Complex covariance structures
- Complicated couplings between fixed and random effects
- Different sources of observations needing different likelihood types
- Standard models are very useful, but should not limit us

Formula interfaces are sometimes frustrating ...

A useful model for longitudinal data:

$$
\begin{aligned}
\mathbf{lnc} &\sim N(\mu, \mathbf{V}), \text{ where} \\
\mu_i &= \mu + \alpha(\texttt{treatm}_i) + \beta(\texttt{month}_i) + \gamma(\texttt{treatm}_i, \texttt{month}_i), \text{ and} \\
V_{i_1,i_2} &= \begin{cases} 0 & , \text{ if } \texttt{cage}_{i_1} \neq \texttt{cage}_{i_2} \text{ and } i_1 \neq i_2 \\ \nu^2 + \tau^2 \exp\left\{ \frac{-(\texttt{month}_{i_1} - \texttt{month}_{i_2})^2}{\rho^2} \right\} & , \text{ if } \texttt{cage}_{i_1} = \texttt{cage}_{i_2} \text{ and } i_1 \neq i_2 \\ \nu^2 + \tau^2 + \sigma^2 & , \text{ if } i_1 = i_2 \end{cases}
\end{aligned}
$$

- This model is implemented by:

```
1 fit.gau <- lme(lnc~month+treatm+month:treatm,
2 random=~1|cage,
3 correlation=corGaus(form=~as.numeric(month)|cage,nugget=TRUE),
4 data=rats)
```

- So many pitfalls and much is hidden. Even difficult to recover model parameters.
- What is τ? Some hours with manual, but re-implement to be sure...
- Restricted by what someone else has put in there. Giant task to move beyond.

Terbuthylazine

- It is a herbicide
- Free terbuthylazine can be washed into the drinking water
- It can be bound to the soil
- Certain bacterias can mineralize it

$$\frac{dB_t}{dt} = -k_1 B_t + k_2 F_t, \qquad B_0 = 0$$
$$\frac{dF_t}{dt} = k_1 B_t - (k_2 + k_3) F_t, \qquad F_0 = 100$$
$$\frac{dM_t}{dt} = k_3 F_t, \qquad M_0 = 0$$

Simplifying

- The system is closed, so $M_t = 100 - B_t - F_t$
- Define $X_t = \begin{pmatrix} B_t \\ F_t \end{pmatrix}$
- The simplified system is:

$$\frac{dX_t}{dt} = \underbrace{\begin{pmatrix} -k_1 & k_2 \\ k_1 & -(k_2 + k_3) \end{pmatrix}}_{A} X_t, \qquad X_0 = \begin{pmatrix} 0 \\ 100 \end{pmatrix}$$

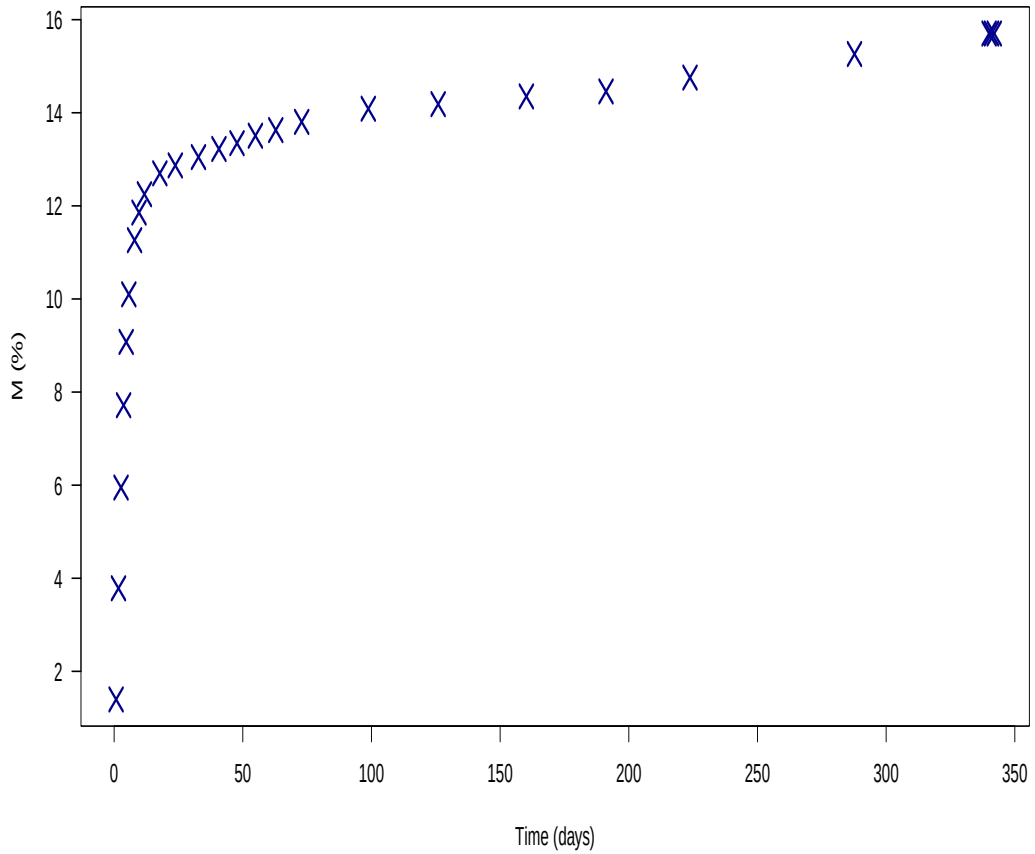
- The system is linear, so it can be be solved via the matrix exponential

$$X_t = e^{At} X_0$$

Observations

- The amount of mineralized terbuthylazine was measured 26 times throughout a year

Time	M
0.77	1.396
1.69	3.784
2.69	5.948
3.67	7.717
4.69	9.077
5.71	10.100
7.94	11.263
9.67	11.856
11.77	12.251
17.77	12.699
23.77	12.869
32.77	13.048
40.73	13.222
47.75	13.347
54.90	13.507
62.81	13.628
72.88	13.804
98.77	14.087
125.92	14.185
160.19	14.351
191.15	14.458
223.78	14.756
287.70	15.262
340.01	15.703
340.95	15.703
342.01	15.703



- The simplest model we can think of would be:

$$M_{t_i} \sim \mathcal{N}\left(100 - \sum X_{t_i}, \sigma^2\right), \quad \text{independent, and with } X_{t_i} = e^{At_i} X_0.$$

```
1  library(Matrix)
2  dat <- read.table('intro/min.dat', skip=3, head=FALSE)
3  nlogL <- function(theta){
4       k <- exp(theta[1:3])
5       sigma <- exp(theta[4])
6       A <- rbind(c(-k[1], k[2]), c(k[1], -(k[2] + k[3])))
7       x0 <- c(0, 100)
8       sol <- function(t) 100 - sum(expm(A*t) %*% x0)
9       pred <- sapply(dat[, 1], sol)
10      -sum(dnorm(dat[, 2], mean = pred, sd = sigma, log = TRUE))
11 }
12 fit <- nlminb(c(-2, -2, -2, -2), nlogL)
13 fit2 <- optim(c(-2, -2, -2, -2), nlogL)
14 fit3 <- nlm(nlogL,c(-2, -2, -2, -2))
15 fit$objective  #gives 0.9392142
16 fit2$value    #gives 19.26905
17 fit3$minimum  #gives 102.766
```

- Difficult because it is non-linear
- Would possibly be helped by accurate gradient info
- Notice this is a very small example with only 4 parameters

What is needed to handle non-standard problems

- Code your log-likelihood
- A good function minimizer
  - I always use `nlminb` guided by accurate gradients

AD aided minimzer

- Want to minimize the negative log likelihood w.r.t. $\theta = (\theta_1, \ldots, \theta_n)$

$$\hat{\theta} = \text{argmin}_\theta \, \ell(\theta | y)$$

- If the dimension of θ is small, any method can be used.
- We would like to handle much larger problems.
- We would like to include latent effects.
- A quasi-Newton minimizer aided by automatic differentiation.

ADMB

- The Newton minimizer is an iterative algorithm.
- It applies the first and second derivatives when iterating to the next step.
- An efficient differentiation procedure is needed.

Automatic differentiation

- We need to calculate $\ell(y|\theta)$ anyway
- The likelihood is in the end a long sequence of simple operations: +,-,*,exp, sin, cos, sqrt, log, ...
- We know how to differentiate each of these
- The chain rule tells us how to combine: $(f(g(x)))' = f'(g(x))g'(x)$
- Automatic differentiation:
  - Keep track of all simple operations when calculating $\ell(y|\theta)$
  - Apply the chain rule to calculate $\ell'(y|\theta)$
- Alternatives:
  - $\ell'(\theta)_i \approx \frac{\ell(\theta+\Delta\theta,x)-\ell(\theta,x)}{\Delta\theta_i}$, Simple but slow and inaccurate
  - Analytical, an excellent option, but difficult for large models.

Template Model Builder:

- TMB developed by Kasper Kristensen (DTU-Aqua)
- Continuously developed since 2009
- ADMB-inspired package
- Applies Laplace approximation for integrating over latent effects
- Automatic sparseness detection
- Combines external libraries: CppAD, Eigen, CHOLMOD
- Parallelism through BLAS
- Parallel user templates
- RTMB: R-interface to set up computational graph
  - TMB: C++ template-based

Example with TMB

- Assume that these 15 numbers follow a negative binomial distribution: 13 5 28 28 15 4 13 4 10 17 11 13 12 17 3
- The TMB code becomes

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

dat = list()
dat$Y = c(13, 5, 28, 28, 15, 4, 13, 4, 10,
      17, 11, 13, 12, 17, 3)

par = list()
par$logsize = 0
par$logitp = 0

obj = MakeADFun(dat, par, DLL="nbin")
opt = nlminb(obj$par, obj$fn, obj$gr)
sdrep = sdreport(obj)
```

nbin.R

```cpp
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(Y);

  PARAMETER(logsize);
  PARAMETER(logitp);

  Type size = exp(logsize);
  Type p=invlogit(logitp);
  Type nll = -sum(dnbinom(Y, size, p, true));
  ADREPORT(size);
  return nll;
}
```

nbin.cpp

Same example via RTMB

- Assume that these 15 numbers follow a negative binomial distribution: 13 5 28 28 15 4 13 4 10 17 11 13 12 17 3
- The RTMB code becomes

```
1  library(RTMB)
2  dat = list()
3  dat$Y = c(13, 5, 28, 28, 15, 4, 13, 4, 10,
       17, 11, 13, 12, 17, 3)
4  par = list()
5  par$logsize = 0
6  par$logitp = 0
7
8  f = function(par){
9    size= exp(par$logsize)
10   phi= plogis(par$logitp)
11   nll= -sum(dnbinom(dat$Y,size,phi,log=TRUE))
12   ADREPORT(size)
13   return(nll)
14 }
15
16 obj = MakeADFun(f,par)
17 opt = nlminb(obj$par,obj$fn,obj$gr)
18 sdrep = sdreport(obj)
```

nbinR.R

- No compilation of C-code needed
  - Makes implementation and debugging much easier!

Standard procedure with RTMB

- Define you negative log-likelihood

```
1 nll = function(par){...}
```

- Set up the AD-machinery:

```
1 obj <- MakeADFun(nll, par)
```

- The negative log liklihood and it's derivative:

```
1 obj$fn() #function value
2 obj$gr() #gradient
```

- Use your favorite optimization algorithm

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

- Calculate uncertainties of parameters and adreport variables

```
1 sdrep = sdreport(obj)
```

Syntax after estimating model

The following command lines are very useful

- Parameter estimates with standard deviation

```
pl = as.list(sdrep,what = "Est")
plsd = as.list(sdrep,what = "Std")
```

- AD-reported variables with standard deviation

```
rl = as.list(sdrep,what = "Est",report = TRUE)
rlsd = as.list(sdrep,what = "Std",report = TRUE)
```

- Show `welcome/linreg.R`

Exercise 1

Assume that the follow 15 numbers follow a negative binomial distribution:

13 5 28 28 15 4 13 4 10 17 11 13 12 17 3

Use the code provided in `nb.R` to:

- Exercise 1a) Find function value and gradient at initial parameter values.
- Exercise 1b) Estimate parameters with maximum likelihood.

```
1  library(RTMB)
2  dat = list()
3  dat$Y = c(13, 5, 28, 28, 15, 4, 13, 4,
       10, 17, 11, 13, 12, 17, 3)
4  par = list()
5  par$logsize = 0
6  par$logitp = 0
7
8  f = function(par){
9    size= exp(par$logsize)
10   phi= plogis(par$logitp)
11   nll= -sum(dnbinom(dat$Y,size,phi,log
         =TRUE))
12   ADREPORT(size)
13   return(nll)
14 }
15
16 obj = ...
17 opt = ...
18 sdrep = ...
```

exercise/nb.R

Gradient based search

```
> opt  = nlminb(obj$par,obj$fn,obj$gr, control = list(trace = 1))
outer mgc:  103.5774
0:     398.42982:  0.00000  0.00000
outer mgc:  112.5499
1:     281.66042: -0.465673 0.884957
outer mgc:  68.03514
2:     205.84484: -0.622067  2.87883
outer mgc:  45.85355
3:     202.84258: -0.821876  2.87009
outer mgc:  1.537242
4:     200.74572: -0.753957  2.68198
outer mgc:  0.6074853
5:     200.68054: -0.753045  2.55830
outer mgc:  0.07071213
6:     200.67113: -0.752270  2.59332
outer mgc:  0.002883933
7:     200.67099: -0.752283  2.58967
outer mgc:  8.665257e-05
8:     200.67099: -0.752282  2.58952
outer mgc:  5.93635e-06
9:     200.67099: -0.752282  2.58952
```

- mgc stands for ”maximum gradient component”
- We introduce inner optimization later

map-functionality in RTMB

- The map-functionality is used to simplify the model by:
  - Fixing parameters to their initial values
  - Coupling parameters

Example with two vectors of negative binomial data:

```
dat$Y = c(13, 5, 28, 28, 15, 4, 13, 4, 10, 17, 11, 13, 12, 17, 3)
dat$Y2 = c(7, 3, 5, 2, 2, 1, 18, 0, 6, 7)
par$logsize = c(0,0)
par$logitp = c(0,0)
f = function(par){
     nll = -sum(dnbinom(dat$Y,exp(par$logsize[1]),plogis(par$logitp[1]),log=TRUE))
     nll = nll -sum(dnbinom(dat$Y2,exp(par$logsize[2]),plogis(par$logitp[2]),log=TRUE))
     return(nll)
}

map = list(logsize= as.factor(c(0,0))) #Couple overdispersion
#map = list(logsize= as.factor(c(NA,NA)))#Set overdispersion to its initial value
obj = MakeADFun(f, par,map = map)
opt = nlminb(obj$par, obj$fn, obj$gr)
```

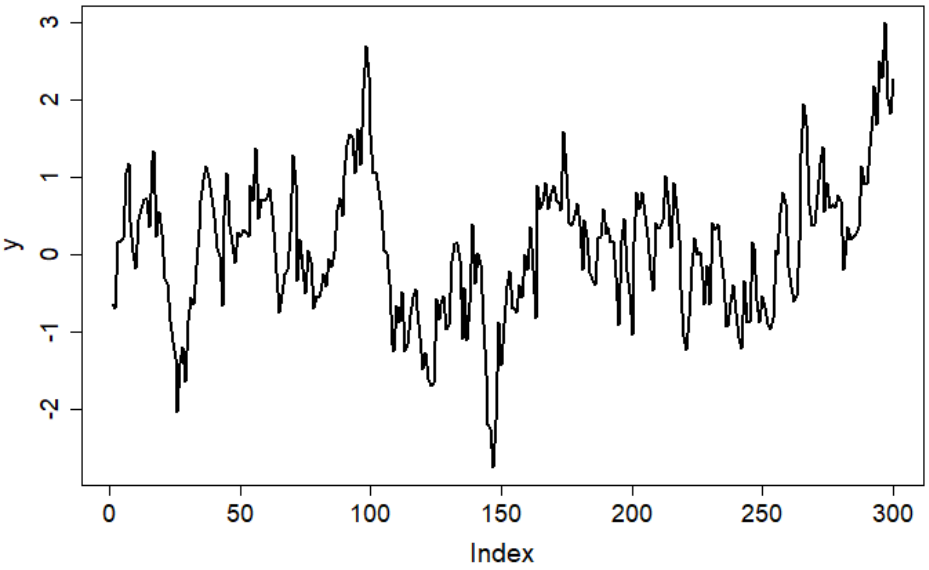
Exercise 2, Observed stationary AR1

Assume $Y \sim N(0, \Sigma_{AR1})$, i.e., that

$$y_1 \sim N(0, \frac{\sigma^2}{1-\rho^2})$$

$$y_i = \rho y_{i-1} + \epsilon_i, \text{ for } 1 < i \leq N$$

and $\epsilon_i \sim N(0, \sigma^2)$.



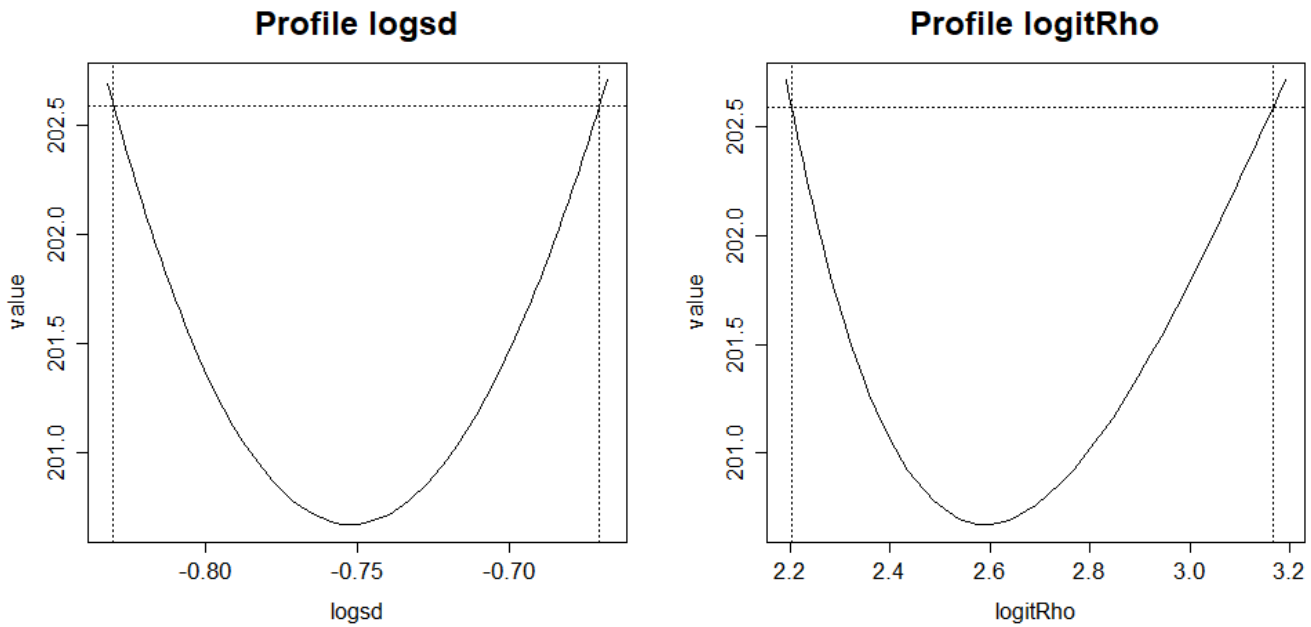
- Exercise 2a: Estimate the model with RTMB using `RTMB::dautoreg`
  - Code to get you started is in `ar1.R`
- Exercise 2b: Use `map` to assume a random walk, i.e., $\rho = 1$. Is ρ significantly different from 1 based on a likelihood ratio test?

Profile

- TMB provides functionality for calculating the profile

```
1 profile = TMB::tmbprofile(obj, "parName")
```

- Profile functions in exercise 2:



- Having problems with a parameter?
  - The profile may provide intuition