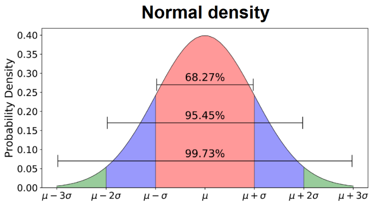
Maximum likelihood estimation

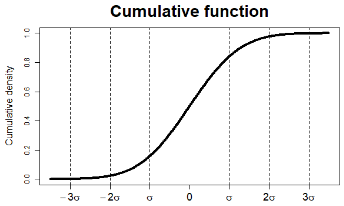
Olav N.R. Breivik

Thanks to Anders Nielsen for borrowing course material

Let's first define the normal distribution:

- Also called the Gaussian distribution
- Two parameters:
  - Expectation μ
  - Standard deviation σ





- Notation: $y \sim N(\mu, \sigma^2)$
- The density is

$$\phi(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

Syntax in R:

```
dnorm(y,mu,sigma) //Density
```

What is a model

Deterministic model:

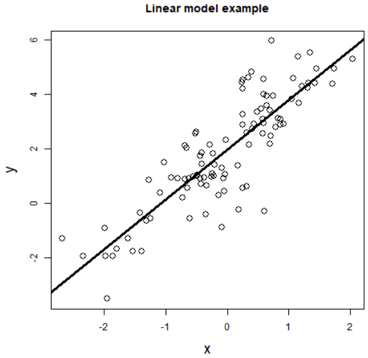
$$y_i = \beta_0 + \beta_1 x_i$$

Statistical model:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

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

- In statistical modeling we pay a loot of attention to ϵ_i



We approximate the real world with a statistical model

- Let **y** be a vector of observations
- Define your statistical model: **y** = model + noise
- Simple linear case: $\mathbf{y} = \beta\mathbf{x} + \epsilon$
- Parametric assessment model:

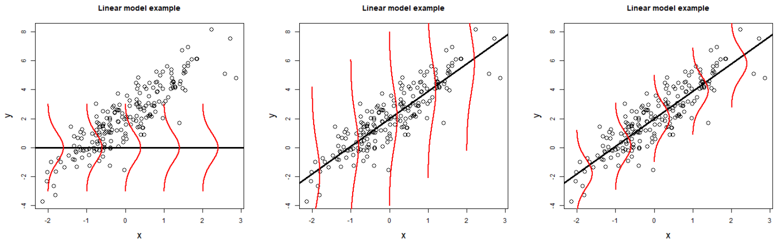
$$\mathbf{y} = f(\boldsymbol{\theta}) + \epsilon$$

  - $\boldsymbol{\theta}$ is model parameters, e.g., recruitment and fishing mortality
- State space assessment model (More on this later!):

$$\mathbf{y} = f(\boldsymbol{\theta}, \mathbf{u}) + \epsilon$$

  - $\boldsymbol{\theta}$ is model parameters, e.g., catchability
  - **u** is random effects; abundance-at-age and fishing mortality
- The data should be of a type expected by the model
  - Chose the parameters that make your model match data.

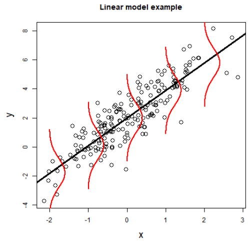
The simple linear example $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$:



- Chose the parameters that make your model match data.

- Log-likelihood is easier to optimize numerically
- Remember $\log\left(\prod_{i=1}^{n} f(x_i)\right) = \sum_{i=1}^{n} \log f(x_i)$
- The simple linear example:

$$
\begin{aligned}
\ell(\boldsymbol{\theta}|y) &= \sum_{i=1}^{n} \log N(y_i, \beta_0 + \beta_1 x_i, \sigma^2) \\
&= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\Big(y_i - (\beta_0 + \beta_1 x_i)\Big)^2
\end{aligned}
$$



Maximum likelihood

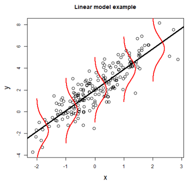
- The data should be of a type expected by the model

In the linear example $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$:

- $\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log N(y_i, \beta_0 + \beta_1 x_i, \sigma^2)$

Find β_0, β_1 and σ that maximize $\ell(\boldsymbol{\theta})$

```r
f = function(par){
     nll = 0
     yhat = par["beta0"] + par["beta1"]*x
     sigma = exp(par["logSigma"])
     for(i in 1:length(y)) nll = nll - dnorm(y[i],yhat[i], sigma,TRUE)
     return(nll)
}
par =list(beta0 = 0,beta1 = 0,logSigma = 3)
fit = nlminb(par, f)
```



Maximum likelihood

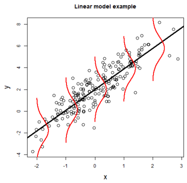
- The data should be of a type expected by the model

In the linear example $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$:

- $\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log N(y_i, \beta_0 + \beta_1 x_i, \sigma^2)$

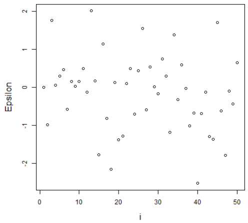
Find β_0, β_1 and σ that maximize $\ell(\boldsymbol{\theta})$

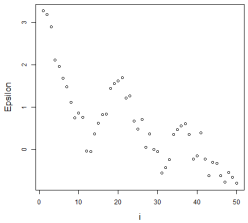
```r
library(RTMB)
f = function(par){
     RTMB::getAll(par,data)
     nll = 0
     yhat = beta0 + beta1*x
     sigma = exp(logSigma)
     for(i in 1:length(y)) nll = nll - dnorm(y[i],yhat[i], sigma,TRUE)
     return(nll)
}
obj = MakeADFun(f,par)
fit = nlminb(par, obj$f, obj$gr)
```



The data should be expected under the model

- What was our assumptions in the example?
- We can verify if our assumptions are violated!
  - More on this later!





Inference on large models

- In these simple example any slow procedure can be used
- We want to be able to do it in much (!) larger models
- Solution: Gradient guided optimization
- Need efficient procedure for calculating the derivative!
- Use automatic differentiation to do the work for you.
    - RTMB does this for you
    - RTMB has many more functionalities that we will cover during the course

What is RTMB

- General tool for conducting inference
- Users write the statistical model in plain R
- RTMB sets up machinery for:
  - Calculating the provided function
  - Marginalizing over latent effects efficiently
  - Differentiate the provided function
- 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:

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

- This model is implemented by:

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

$$\begin{tikzcd}
\boxed{B} \arrow[r, shift left, "k_1"] & \boxed{F} \arrow[l, shift left, "k_2"] \arrow[r, "k_3"] & \boxed{M}
\end{tikzcd}$$

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

Simplifying

$$\begin{tikzcd}
\boxed{B} \arrow[r, shift left, "k_1"] & \boxed{F} \arrow[l, shift left, "k_2"] \arrow[r, "k_3"] & \boxed{M}
\end{tikzcd}$$

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

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

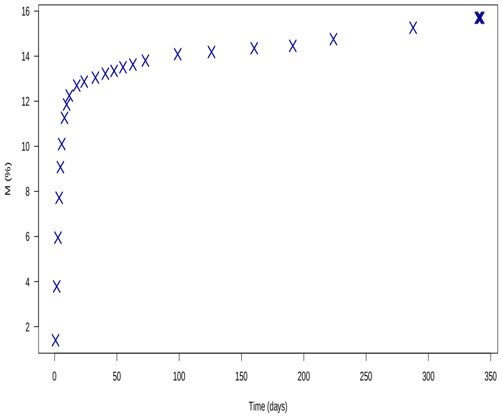
- The system is linear, so it can be be solved via the matrix exponential

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

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:

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

```r
library(Matrix)
dat <- read.table('intro/min.dat', skip=3, head=FALSE)
nlogL <- function(theta){
     k <- exp(theta[1:3])
     sigma <- exp(theta[4])
     A <- rbind(c(-k[1], k[2]), c(k[1], -(k[2] + k[3])))
     x0 <- c(0, 100)
     sol <- function(t) 100 - sum(expm(A*t) % *% x0)
     pred <- sapply(dat[, 1], sol)
     -sum(dnorm(dat[, 2], mean = pred, sd = sigma, log = TRUE))
}
fit <- nlminb(c(-2, -2, -2, -2), nlogL)
fit2 <- optim(c(-2, -2, -2, -2), nlogL)
fit3 <- nlm(nlogL,c(-2, -2, -2, -2))
fit$objective  #gives 0.9392142
fit2$value     #gives 19.26905
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. $\theta = (\theta_1, \dots, \theta_n)$

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

- If the dimension of θ is small, computation time is no problem.
- We would like to handle much larger problems.
- We would like to include latent effects.
- A quasi-Newton minimizer aided by automatic differentiation.

ADMB

ADMB

Automatic Differentiation Model Builder

- 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 11 numbers follow a normal distribution: 0.8, -1.0, -0.7, -2.0, -2.1, 0.6, 0.9, 0.1, 1.8, 4.1, -1.0
- The TMB code becomes

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

dat = list(Y = c(0.8,-1.0,-0.7,-2.0,-2.1,
      0.6,0.9,0.1,1.8,4.1,-1.0))
par = list(mu = 0,
           logsd = 0)

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

normal.R

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

  PARAMETER(mu);
  PARAMETER(logsd);

  Type sd = exp(logsd);
  Type nll = -sum(dnorm(Y, mu, sd, true));
  return nll;
}
```

normal.cpp

Same example via RTMB

- Assume that these 11 numbers follow a normal distribution: 0.8, -1.0, -0.7, -2.0, -2.1, 0.6, 0.9, 0.1, 1.8, 4.1, -1.0
- The RTMB code becomes

```r
library(RTMB)

dat = list(Y = c(0.8,-1.0,-0.7,-2.0,-2.1,
      0.6,0.9,0.1,1.8,4.1,-1.0))
par = list(mu = 0,
           logsd = 0)

f = function(par){
  getAll(dat,par)
  sd= exp(logsd)
  nll= -sum(dnorm(Y,mu,sd,log=TRUE))
  return(nll)
}

obj = MakeADFun(f,par)
opt = nlminb(obj$par,obj$fn,obj$gr)
sdrep = sdreport(obj)
```

normalR.R

☺

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

Standard procedure with RTMB

- Define you negative log-likelihood

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

- Set up the AD-machinery:

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

- The negative log liklihood and it's derivative:

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

- Use your favorite optimization algorithm

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

- Calculate uncertainties of parameters and adreport variables

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

Syntax after estimating model

The following command lines are very useful

- Parameter estimates with standard deviation

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

- AD-reported variables with standard deviation

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

Exercise 1

Assume that these 11 numbers follow a normal distribution: 0.8, -1.0, -0.7, -2.0, -2.1, 0.6, 0.9, 0.1, 1.8, 4.1, -1.0

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

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

```r
library(RTMB)

dat = list(Y = c(0.8,-1.0,-0.7,
      -2.0,-2.1,0.6,0.9,0.1,
      1.8,4.1,-1.0))
par = list(mu = 0,
           logsd = 0)

f = function(par){
  getAll(dat,par)
  sd= exp(logsd)
  nll= -sum(dnorm(Y,mu,sd,log=TRUE))
  return(nll)
}

obj = MakeADFun(f,par)
opt = nlminb(obj$par,obj$fn,obj$gr,
      control = list(trace = 1))
```

exercise/normalR.R

Gradient based search

```
> opt = nlminb(obj$par,obj$fn,obj$gr, control = list(trace = 1))
outer mgc:  21.77
0:     26.493324:  0.00000  0.00000
outer mgc:  6.565006
1:     23.299802: 0.0687392 0.997635
outer mgc:  3.396411
2:     22.122361: 0.477263 0.746568
outer mgc:  1.673685
3:     21.640757: 0.0839206 0.472326
outer mgc:  0.4877506
4:     21.583968: 0.154481 0.565411
outer mgc:  0.03740074
5:     21.577830: 0.138927 0.544382
outer mgc:  0.003814453
6:     21.577788: 0.135337 0.542745
outer mgc:  0.003364048
7:     21.577787: 0.136779 0.542526
outer mgc:  0.0001571183
8:     21.577786: 0.136370 0.542686
outer mgc:  8.291266e-06
9:     21.577786: 0.136363 0.542678
```

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

Parameter transformation

For parameters with natural bounds we can transform them to the interval of interest:

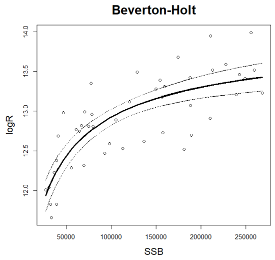
1. only positive
   - $\theta = e^{\alpha}$, where $\alpha \in \mathbb{R}$
2. between -1 and 1
   - $\theta = 2/(1 + e^{-\alpha}) - 1$, where $\alpha \in \mathbb{R}$
3. Increasing positive vector
   -
   - $\theta = (e^{\alpha_1}, e^{\alpha_1} + e^{\alpha_2}, ..., e^{\alpha_1} + \cdots + e^{\alpha_n})$, where $\alpha_1, ..., \alpha_n \in \mathbb{R}$

Exercise 2: Beverton-Holt stock recruitment model

- The Beverton-Holt model can be written (slightly re-parametrized) as:

$$\log R_i = \log(a) + \log(\mathrm{ssb}_i) - \log(1 + \exp(\log(b))\mathrm{ssb}_i) + \varepsilon_i,$$

  where $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$
- A data set of SSB and $\log(R)$ is provided in `bh.dat`
- Exercise 2: Estimate the model parameters a and b and σ.
  - To get you started, see `bhEx.R`



map-functionality in RTMB

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

Example with fixing variance parameter in previous exercise:

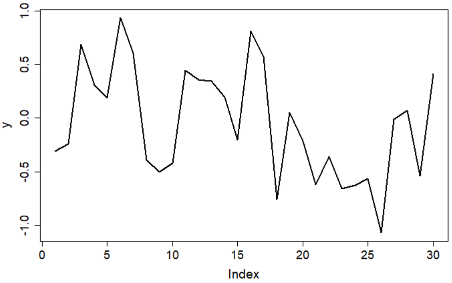
```
map = list()
map$logSigma = as.factor(NA)
par$logSigma = log(0.5)
obj <- MakeADFun(f,par,map = map)
opt <- nlminb(obj$par,obj$fn,obj$gr)
```

- Now $\sigma = 0.5$ and it is not estimated

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 3a: The model is implemented in `ar1.R`. Read trough it and make sure you understand the implementation.
- Exercise 3b: Use `map` to assume independence between the y's, i.e., $\rho = 0$.

Motivation: In SAM we will assume AR1 structure in observations and in fishing mortality increments.

In plenary, illustrate:

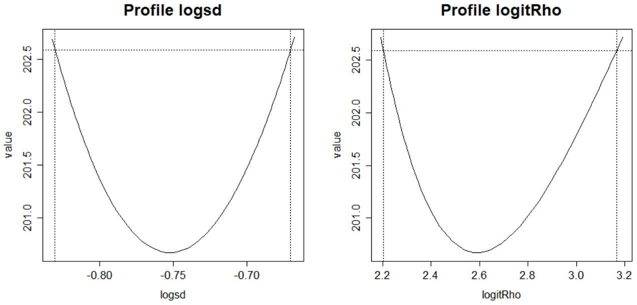
- AIC difference.
- Is the reduction in AIC significant?

Profile

- TMB provides functionality for calculating the profile

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

- Profile functions in exercise 3:



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