Summary from last year

Anders Nielsen & Ethan Lawler
an@aqua.dtu.dk

Last year we talked about

- Introduction and installation
- **Running a first example**
- Maximum likelihood in TMB (quick reminder: estimator, uncertainty, test)
- Getting data in and results out
- Nonlinear model example
- Dealing with parameters (bounds, phases, transformations, mapping)
- Parametric age-based stock assessment model
- Uncertainty quantification (delta-method, profile, MCMC, (bias-correction))
- Splines in TMB
- Biomass dynamic model
- Debugging TMB models
- **Random effects in TMB**
- Simple state-space models
- **Assessment as state-space models**
- Model validation in state-space models
- Simulation within the TMB code
- Checking the Laplace approximation
- Parallel computations

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:

```
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.

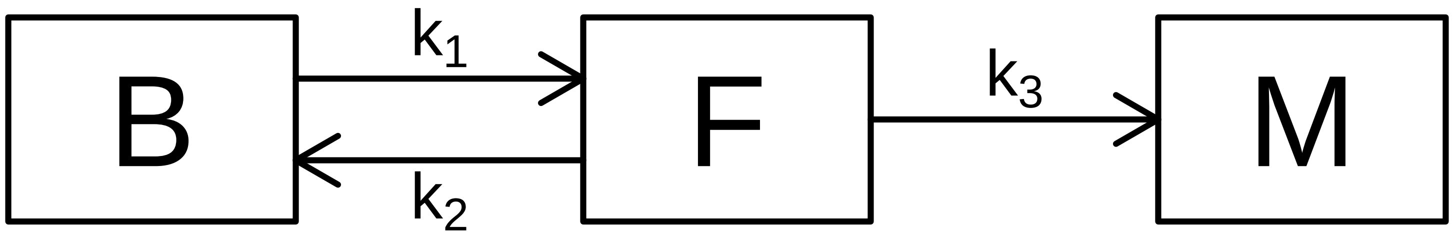
Terbuthylazine

- 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



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

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

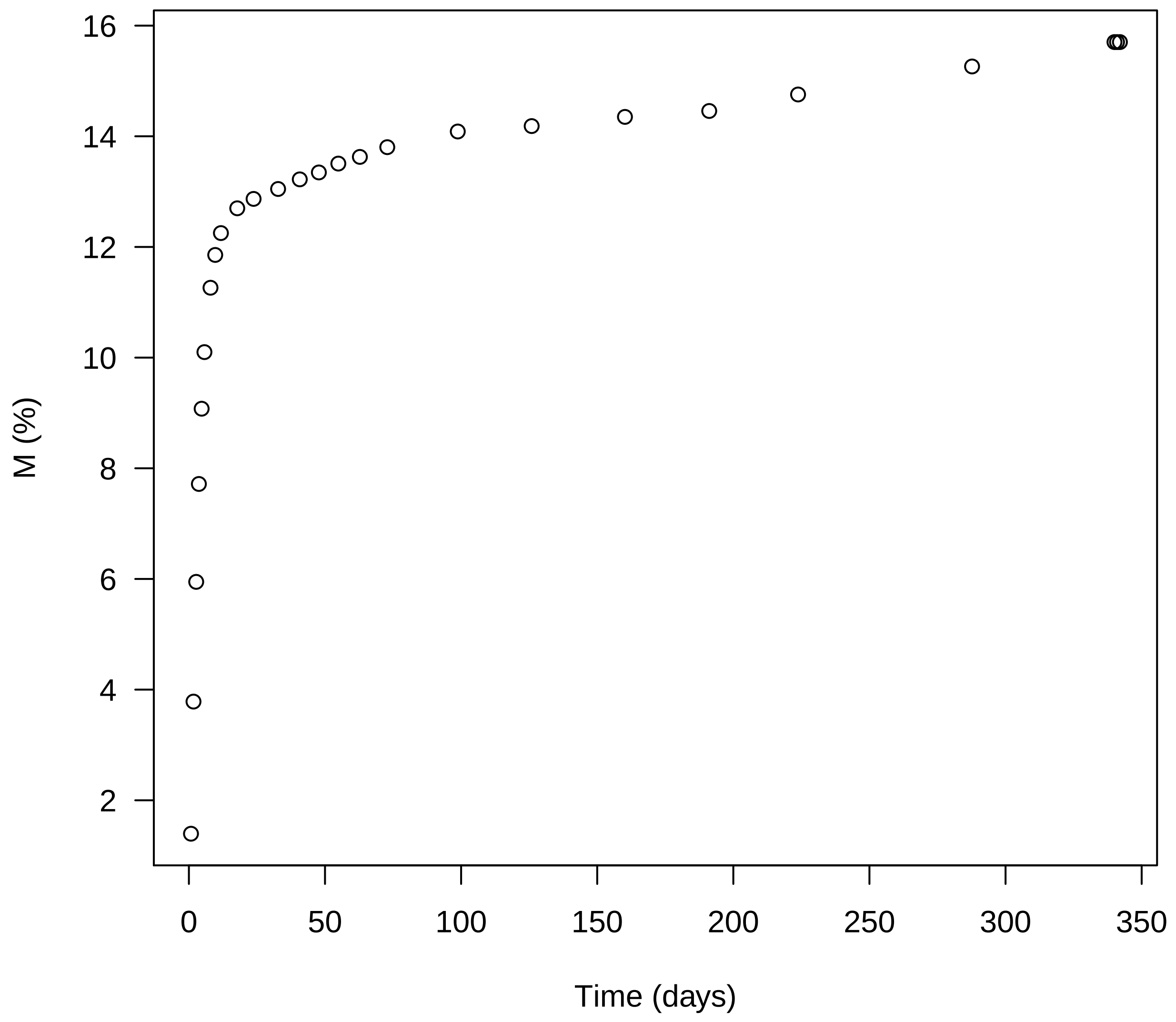
- The system is linear, so it can be be solved for instance via the matrix exponential

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

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



Simplest statistical model

- The simplest model we can think of would be:

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

```
> library(Matrix)
> 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))
+ }
> system.time(fit <- optim(c(-2, -2, -2, -2), nlogL, hessian = TRUE))

   user  system elapsed
 19.409   0.004  19.428

> fit$value

[1] 19.26905

> fit$convergence

[1] 0
```

- Try some of the different minimizers in R

```
> library(optimx)
> fit<-optimx(c(-2,-2,-2,-2),nlogL,hessian=TRUE,control=list(all.methods=TRUE))
> fit
```

```
  fvalues      method fns grs conv  KKT1  KKT2 xtimes
 153.3056      bobyqa 144  NA    0  TRUE FALSE  9.629
 102.7661      Rcgmin  85  50    0  TRUE FALSE 22.314
 102.7660         nlm  NA  NA    0  TRUE FALSE 11.865
 102.7660        BFGS  79  18    0  TRUE FALSE 15.005
 102.7660      Rvmmin  81  15    0  TRUE FALSE 10.517
 102.7660          CG 567 101    1  TRUE FALSE 91.569
 91.17466      newuoa 696  NA    0  TRUE FALSE 46.063
 19.26905 Nelder-Mead 223  NA    0 FALSE FALSE 14.837
0.9392184      ucminf  40  40    0 FALSE  TRUE 14.953
0.9392142         spg 198  NA    0 FALSE  TRUE 55.732
0.9392142    L-BFGS-B  85  85    0 FALSE  TRUE 50.807
0.9392142      nlminb  33 128    0  TRUE  TRUE 10.729
```

- Difficult because it is non-linear
- Would possibly be helped by accurate gradient info
- Runs in a fraction of a second in TMB (exercise)
- Notice this is a miniature example with only 4 parameters

What is needed to handle a non-standard problem

- A purely parametric assessment model has more than 100 model parameters and it is non-linear
- Code up the negative log likelihood function
- **A good function minimizer**

AD aided minimizer

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

$$\widehat{\theta} = \operatorname*{argmin}_{\theta} \ell(y|\theta)$$

- If the dimension of θ is low (say n less than 5) any method can be used (grid search, random search, finite difference approximations, ...)
- We would like to be able to handle much larger problems
- Important for fixed effects models, and even more for random effects models
- A quasi-Newton minimizer aided by automatic differentiation

Quasi-Newton minimizer

ADMB

Automatic Differentiation Model Builder

- A Newton minimizer is an iterative algorithm
- Each step assumes that the function $\ell(x, \theta)$ can be approximated locally by a quadratic function
- It uses the first ℓ'_θ and second ℓ''_θ derivatives to find the minimum
- Instead of calculating ℓ''_θ at every step, a quasi-Newton minimizer uses successive first derivatives ℓ'_θ to approximate ℓ''_θ
- So a fast and accurate way to calculate ℓ'_θ is needed

Automatic Differentiation

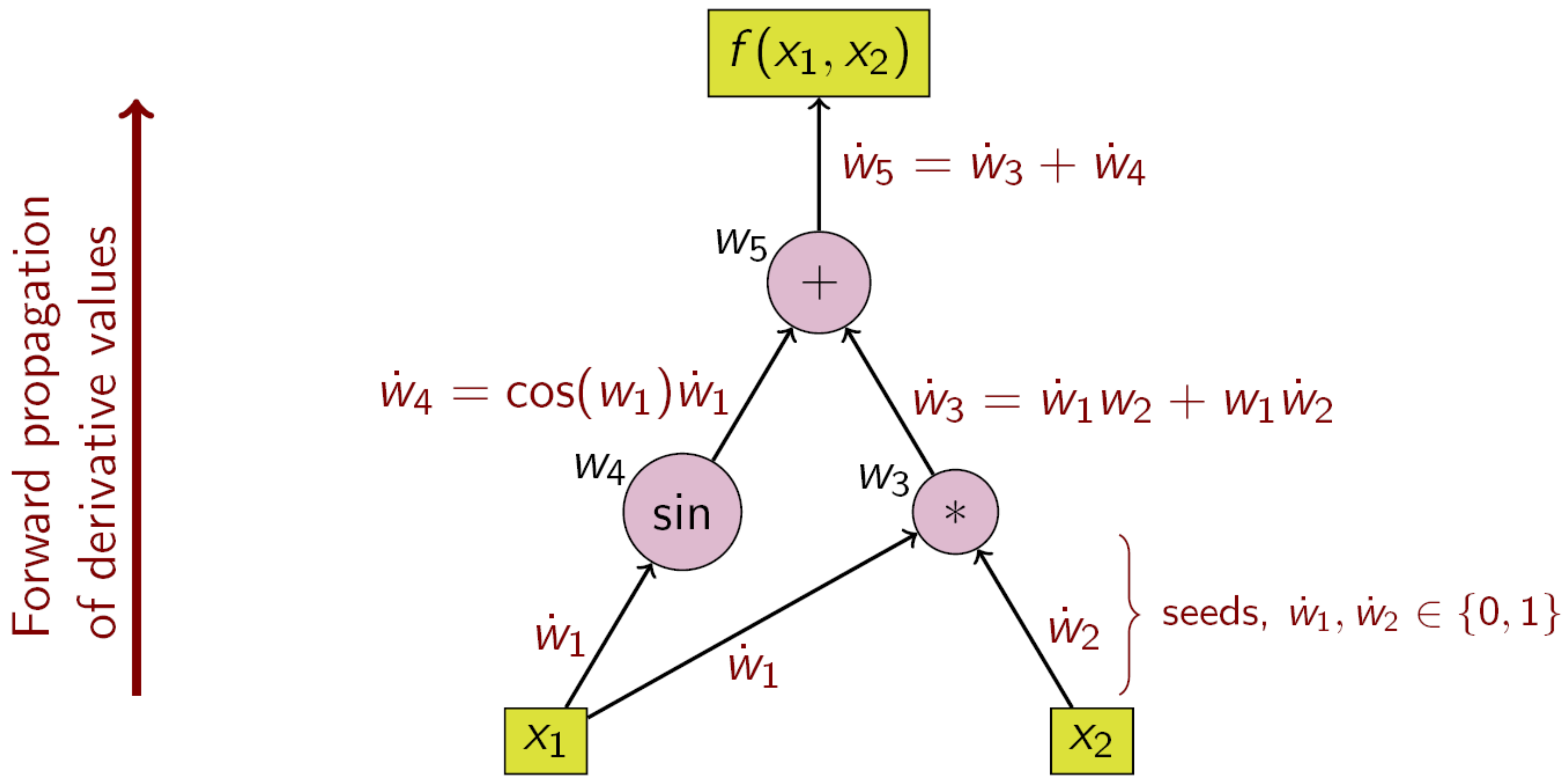
- We need to write a program to compute $\ell(\theta, x)$ anyway
- A computer program is a long list of simple operations:
  '+', '-', '*', '/', 'exp', 'log', 'sin', 'cos', 'tan', 'sqrt', and so on
- We know how to derive each of these operations
- The chain rule tells us how to combine: $(f(g(x)))' = f'(g(x))g'(x)$
- So if the computer is instructed to:
  - keep track of all the simple operations used when calculating $\ell(\theta, x)$
  - use the simple derivative formulas and the chain rule
- Then once $\ell(\theta, x)$ is computed, we also have ℓ'_θ with a minimum of extra calculations
- This is fast and accurate, and the difficult part is built into TMB(!)
- Alternatives:
  — Finite difference: $(\ell'_\theta)_i \approx \frac{\ell(\theta_i+\Delta\theta_i,x)-\ell(\theta,x)}{\Delta\theta_i}$ Simple, but slow and inaccurate
  — Analytical: Excellent option, but difficult in larger models

```cpp
#include <math.h>
#include <iostream.h>

class result {
  private: double v,d;
  public: result(){v = 0;d= 0;};
          result(double val){v = val; d = 0;};
          result(double val,double der){v = val; d = der;};
          double Value(){return v;};
          double Der(){return d;};
};

class parameter: public result {
  public: parameter(double pval) : result(pval,1.0) {};
          parameter() : result(0.0,1.0) {};
};

result sin(result n){
  return result(sin(n.Value()), cos(n.Value())*n.Der());
};

result operator*(result n1,result n2){
  return(result(n1.Value()*n2.Value(), n1.Der()*n2.Value() + n2.Der()*n1.Value()));
};

ostream& operator<<(ostream& o,result n){
  o << n.Value() << " (Derative: " << n.Der() << ") ";
  return o;
}

int main(int argc, char* argv[]){
  parameter theta(2);
  result y;
  y = sin(theta*theta);
  cout << "The result is " << y << endl;
}
```

cpp/ad.cpp

```
The result is -0.756802 (Derivative: -2.61457)
```

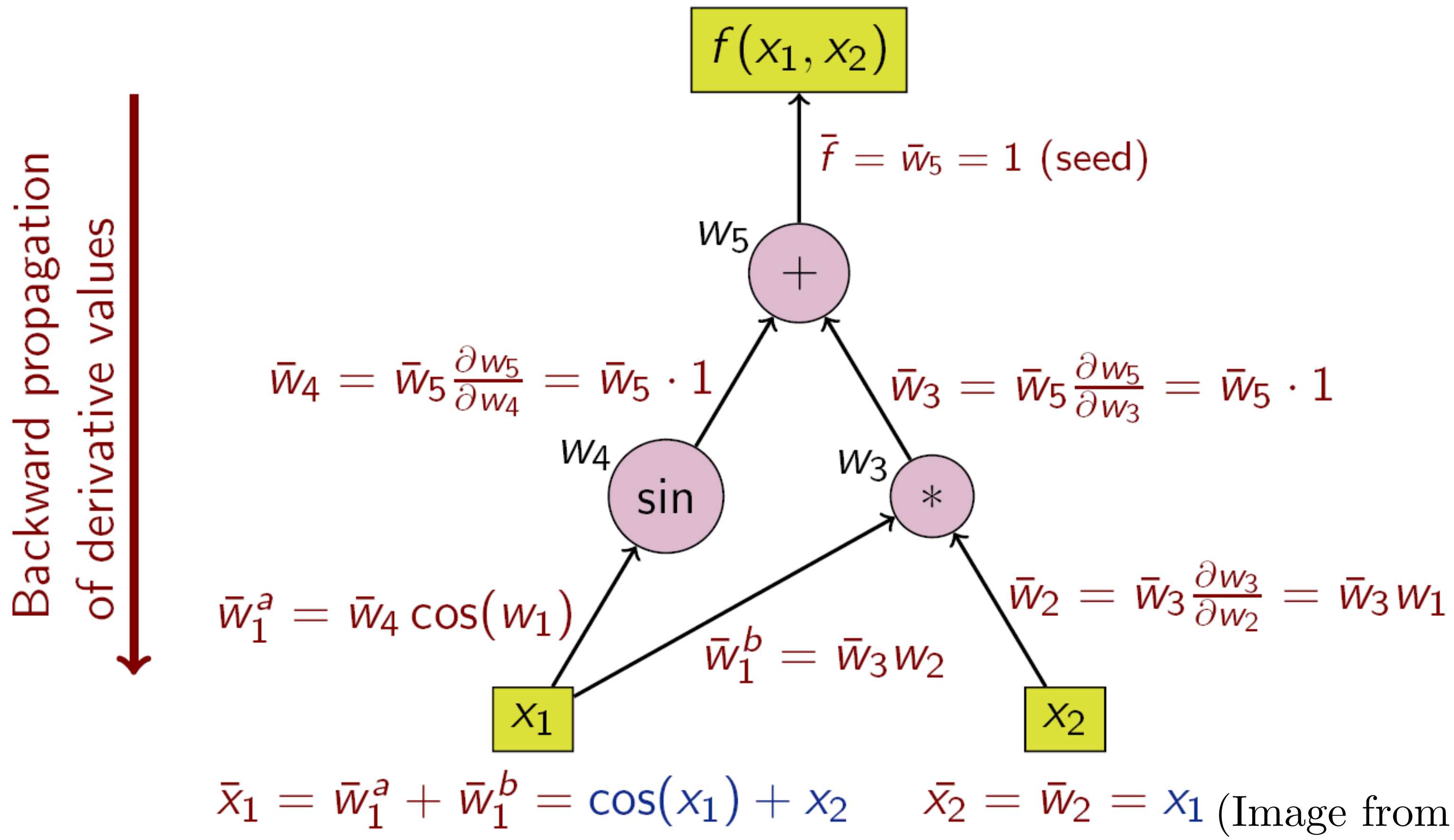
Forward and reverse mode



Forward propagation of derivative values
$f(x_1, x_2)$
$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$
w_5
$+$
$\dot{w}_4 = \cos(w_1)\dot{w}_1$
$\dot{w}_3 = \dot{w}_1 w_2 + w_1 \dot{w}_2$
w_4
sin
w_3
$*$
seeds, $\dot{w}_1, \dot{w}_2 \in \{0, 1\}$
$\dot{w}_1$
$\dot{w}_1$
$\dot{w}_2$
x_1
x_2

(Image from Wikipedia)

- Forward mode is easy to understand and implement
- Not efficient when θ is high dimensional


Backward propagation of derivative values
$f(x_1, x_2)$
$\bar{f} = \bar{w}_5 = 1$ (seed)
w_5
$+$
$\bar{w}_4 = \bar{w}_5 \frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \bar{w}_5 \frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
$*$
$\bar{w}_1^a = \bar{w}_4 \cos(w_1)$
$\bar{w}_2 = \bar{w}_3 \frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
$\bar{w}_1^b = \bar{w}_3 w_2$
x_1
x_2
$\bar{x}_1 = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \bar{w}_2 = x_1$
(Image from Wikipedia)

- Requires recording a stack of all operations
- Efficient in number of operations ($C(\ell'_\theta) < 4C(\ell)$ [a])
- TMB uses reverse mode
- Except for random effects models where a combo of forward and reverse mode is used

[a] Griewank, A., 2000. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia, PA.

Template Model Builder (TMB):

- Developered by Kasper Kristensen (DTU-Aqua)
- ADMB inspired R-package
- Combines external libraries: CppAD, Eigen, CHOLMOD
- Continuously developed since 2009
- Implements Laplace approximation for random effects
- C++ Template based
- Automatic sparseness detection
- Parallelism through BLAS
- Parallel user templates
- Parallelism through `parallel` package

Example

- 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

```cpp
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator
    () ()
{
  DATA_VECTOR(Y);
  PARAMETER(logsize);
  PARAMETER(p);
  Type size = exp(logsize);
  Type nll = -sum(dnbinom(Y, size, p,
      true));
  ADREPORT(size);
  return nll;
}
```

nbin.cpp

```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$logsize <- 0
par$p <- 0.5

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

nbin.R

Random effects

Example: Paired observations

- Two methods A and B to measure blood cell count (to check for the use of doping).
- Paired study.

Person ID	**Method A**	**Method B**
1	5.5	5.4
2	4.4	4.9
3	4.6	4.5
4	5.4	4.9
5	7.6	7.2
6	5.9	5.5
7	6.1	6.1
8	7.8	7.5
9	6.7	6.3
10	4.7	4.2

- It must be expected that two measurements from the same person are correlated, so a paired t-test is the correct analysis
- The t-test gives a p-value of 5.1%, which is a borderline result...
- But more data is available

- In addition to the planned study 10 persons were measured with only one method
- Want to use all data, which is possible with random effects
- Assume these 20 are ramdomly selected from a population where the blod cell count is normally distributed
- Consider the following model:
  $C_i = \alpha(M_i) + B(P_i) + \varepsilon_i, \quad i = 1 \dots 30$
  $\alpha(M_i)$ the 2 fixed method effects
  $B(P_i) \sim \mathcal{N}(0, \sigma_P^2)$ the 20 random effects
  $\varepsilon_i \sim \mathcal{N}(0, \sigma_R^2)$ measurement noise
  All $B(P_i)$ and ε_i are assumed independent
- This model uses all data and gives a 95% c. i. for the method bias $\alpha(A) - \alpha(B)$ which is: $(0.04; 0.41)$.
- Notice that now there is a (slightly) significant method bias.

Person ID	**Method A**	**Method B**
1	5.5	5.4
2	4.4	4.9
3	4.6	4.5
4	5.4	4.9
5	7.6	7.2
6	5.9	5.5
7	6.1	6.1
8	7.8	7.5
9	6.7	6.3
10	4.7	4.2
11		5.1
12		4.4
13		4.5
14		5.3
15		7.5
16	5.7	
17	6.0	
18	7.5	
19	6.5	
20	4.2	

Random effect model

- In purely fixed effects models we have
  - Random variables we observe
  - Model parameters we want to estimate
- In random effects models we have
  - Random variables we observe
  - Random variables we do NOT observe
  - Model parameters we want to estimate
- This model class is very useful and goes by many names: random effects models, mixed models, latent variable models, state-space models, frailty models, hierarchical models, ...

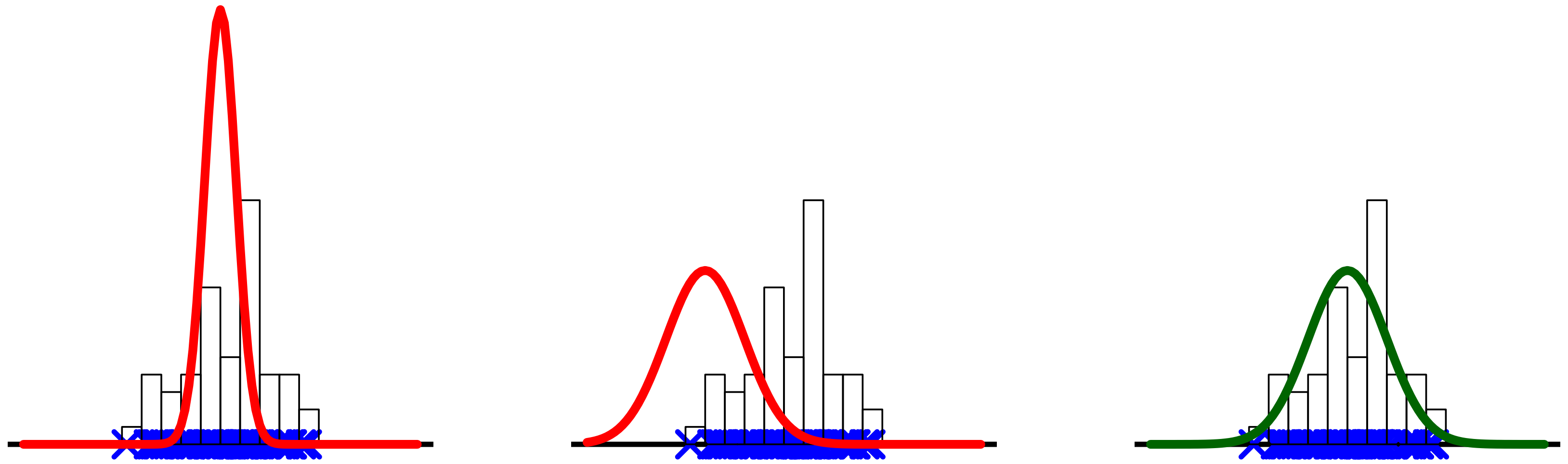
So the difference is that we have some quantity with a distribution, which is unobserved.

Reminder: Estimation with purely fixed effects

- We have:

  Observations: $y = (y_1, y_2, \ldots, y_n)$

  Parameters (μ, σ) in model: $y_i \sim N(\mu, \sigma^2)$



- Choose parameters which makes our model best match the data (optimize likelihood).

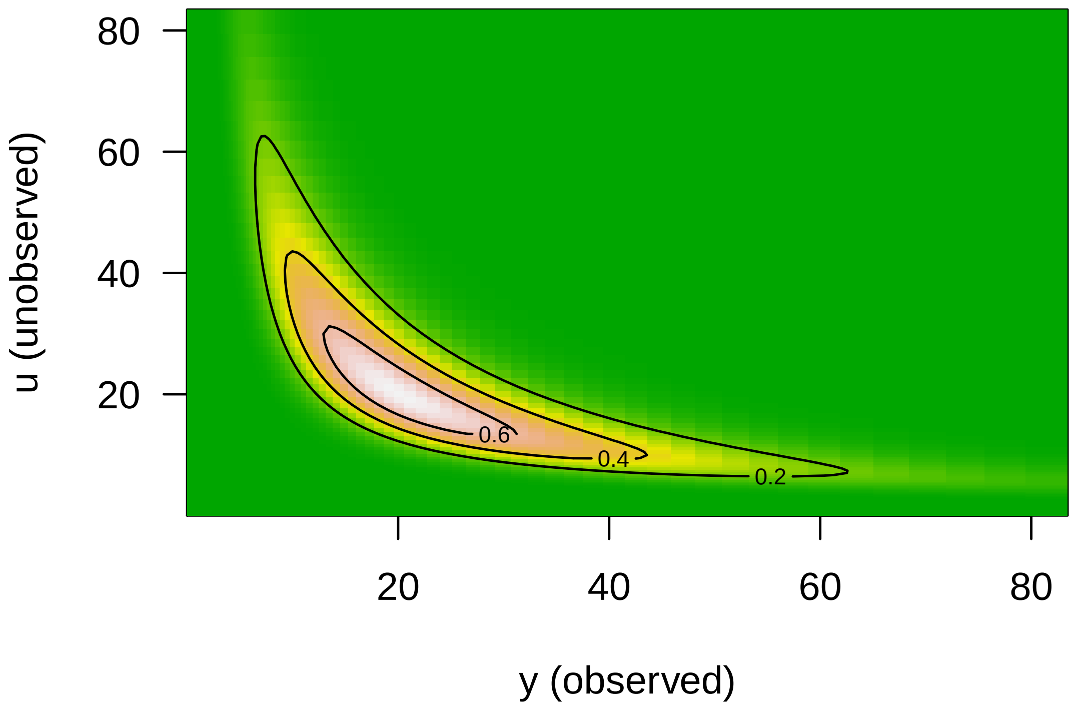
Estimation with random effects

- We have:

  Observations: y

  Unobserved random effects: u

  Parameters (θ) in model: $(y, u) \sim D(\theta)$



- How do we estimate our parameters when some of our observations are not observed?

Estimation with random effects — 2

- The banana is only an intermediate calculation

  1: Joint model (banana) is determined from model parameters θ

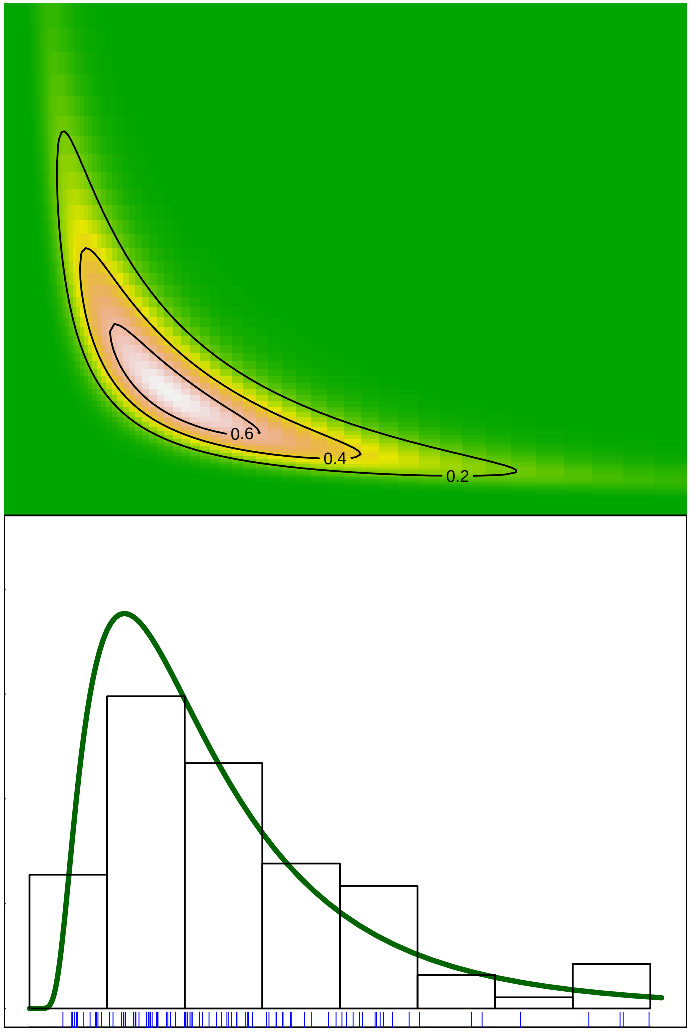
  2: Marginal model is calculated from joint by integration

  3: Marginal is matched to data as always

- Imagine the distribution $D(\theta)$ is described by a likelihood function $L(y, u, \theta)$, then:

  $$L_M(y, \theta) = \int L(y, u, \theta) du$$

  is the marginal likelihood.



Examples where this is the underlying technique

- Random effects are often used:

  Time series models via state-space models Correlation via hidden processes

  Split-plot models Correlated observations within plot

  Repeated measurements Correlated observations within subject

  Over dispersion in Poisson via Negative binomial Patchiness accounted for

  Spatial models Correlation introduced via hidden field

  ...

- Often our only option!

- When something unobserved gives us <u>extra variation</u> or <u>correlated observations</u>.

The Laplace approximation

Reminder: Multivariate normal distribution

The density for a k-dimensional multivariate normal distribution with mean vector μ and covariance matrix Σ is:

$$L(z) = \frac{1}{(2\pi)^{k/2}\sqrt{|\Sigma|}} \exp\left[-\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right]$$

We write $Z \sim N_k(\mu, \Sigma)$.

The log is:

$$\ell(z) = -\frac{1}{2}\left\{k\log(2\pi|\Sigma|) + (z-\mu)^T\Sigma^{-1}(z-\mu)\right\}$$

General random effects models

The general random effects model can be represented by its likelihood function:

$$L_M(\theta; y) = \int_{\mathbb{R}^q} L(\theta; u, y) du$$

– y is the observed random variables

– u is the q unobserved random variables

– θ is the model parameters to be estimated

The likelihood function L is the joint likelihood of both the observed and the unobserved random variables.

The likelihood function for estimating θ is the marginal likelihood L_M obtained by integrating out the unobserved random variables.

Notice we have already seen this

- In the Poisson distribution the variance is equal to the mean, which is an assumption that is not always valid.
- Consider the model:

$$Y \sim \text{Pois}(\lambda), \quad \text{where} \quad \lambda \sim \Gamma\left(n, \frac{1-\phi}{\phi}\right) \quad 0 < \phi < 1$$

- It can be shown that:

$$Y \sim \text{Nbinom}(n, \phi)$$

- Notice:
  - No λ in marginal likelihood for Y
  - Analytical integration is not the typical case

General random effects models

- The integral shown on the previous slide is generally difficult to solve if the number of unobserved random variables is more than a few, i.e. for large values of q.
- A large value of q significantly increases the computational demands due to the product rule which states that if an integral is sampled in m points per dimension to evaluate it, the total number of samples needed is m^q, which rapidly becomes infeasible even for a limited number of random effects.
- The likelihood function gives a very broad definition of mixed models: the only requirement for using mixed modeling is to define a joint likelihood function for the model of interest.
- In this way mixed modeling can be applied to any likelihood based statistical modeling.
- Examples of applications are linear mixed models (LMM) and nonlinear mixed models (NLMM), generalized linear mixed models, but also models based on Markov chains, or SDEs.

The Laplace approximation

- We need to calculate the difficult integral

$$L_M(\theta, y) = \int_{\mathbb{R}^q} L(\theta, u, y) du$$

- So we set up an approximation of $\ell(\theta, u, y) = \log L(\theta, u, y)$

$$\ell(\theta, u, y) \approx \ell(\theta, \hat{u}_\theta, y) - \frac{1}{2}(u - \hat{u}_\theta)^t \left(-\ell''_{uu}(\theta, u, y)|_{u=\hat{u}_\theta}\right)(u - \hat{u}_\theta)$$

- Which (for given θ) is the 2. order Taylor approximation around:

$$\hat{u}_\theta = \underset{u}{\operatorname{argmax}}\; L(\theta, u, y)$$

- With this approximation we can calculate:

$$\begin{aligned}
L_M(\theta, y) &= \int_{\mathbb{R}^q} L(\theta, u, y) du \\
&\approx \int_{\mathbb{R}^q} e^{\ell(\theta, \hat{u}_\theta, y) - \frac{1}{2}(u-\hat{u}_\theta)^t \left(-\ell''_{uu}(\theta,u,y)|_{u=\hat{u}_\theta}\right)(u-\hat{u}_\theta)} du \\
&= L(\theta, \hat{u}_\theta, y) \int_{\mathbb{R}^q} e^{-\frac{1}{2}(u-\hat{u}_\theta)^t \left(-\ell''_{uu}(\theta,u,y)|_{u=\hat{u}_\theta}\right)(u-\hat{u}_\theta)} du \\
&= L(\theta, \hat{u}_\theta, y) \sqrt{\frac{(2\pi)^q}{|\left(-\ell''_{uu}(\theta, u, y)|_{u=\hat{u}_\theta}\right)|}}
\end{aligned}$$

- In the last step we remember the normalizing constant for a multivariate normal, and that $|A^{-1}| = 1/|A|$.
- Taking the logarithm we get:

$$\ell_M(\theta, y) \approx \ell(\theta, \hat{u}_\theta, y) - \frac{1}{2} \log(|\left(-\ell''_{uu}(\theta, u, y)|_{u=\hat{u}_\theta}\right)|) + \frac{q}{2}\log(2\pi)$$

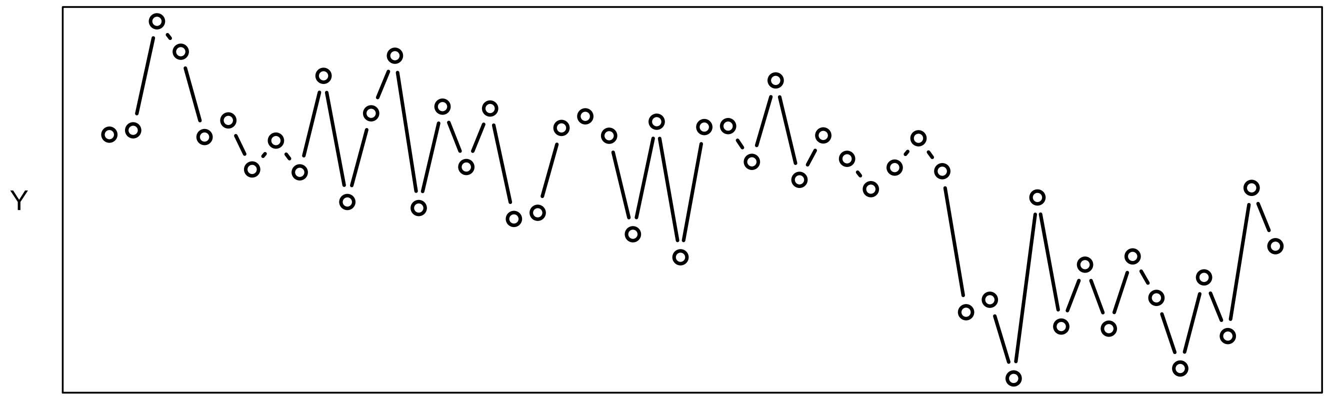
The Laplace approximation

- The Laplace likelihood only approximates the marginal likelihood for mixed models with nonlinear random effects and thus maximizing the Laplace likelihood will result in some amount of error in the resulting estimates.
- It can be shown that joint log-likelihood converges to a quadratic function of the random effect for increasing number of observations per random effect and thus that the Laplace approximation is asymptotically exact.
- In practical applications the accuracy of the Laplace approximation may still be of concern, but often improved numerical approximation of the marginal likelihood (such as Gaussian quadrature) may easily be computationally infeasible to perform.
- Another option for improving the accuracy is importance sampling.

Simple state-space model

State-space model (RW+noise)

- Observation vector Y generated from:
  - $\lambda_i = \lambda_{i-1} + \eta_i$
  - $Y_i = \lambda_i + \varepsilon_i$
  - where $i = 1 \ldots 50$, $\eta_i \sim N(0, \sigma^2_\lambda)$, and $\varepsilon_i \sim N(0, \sigma^2_Y)$ all independent.



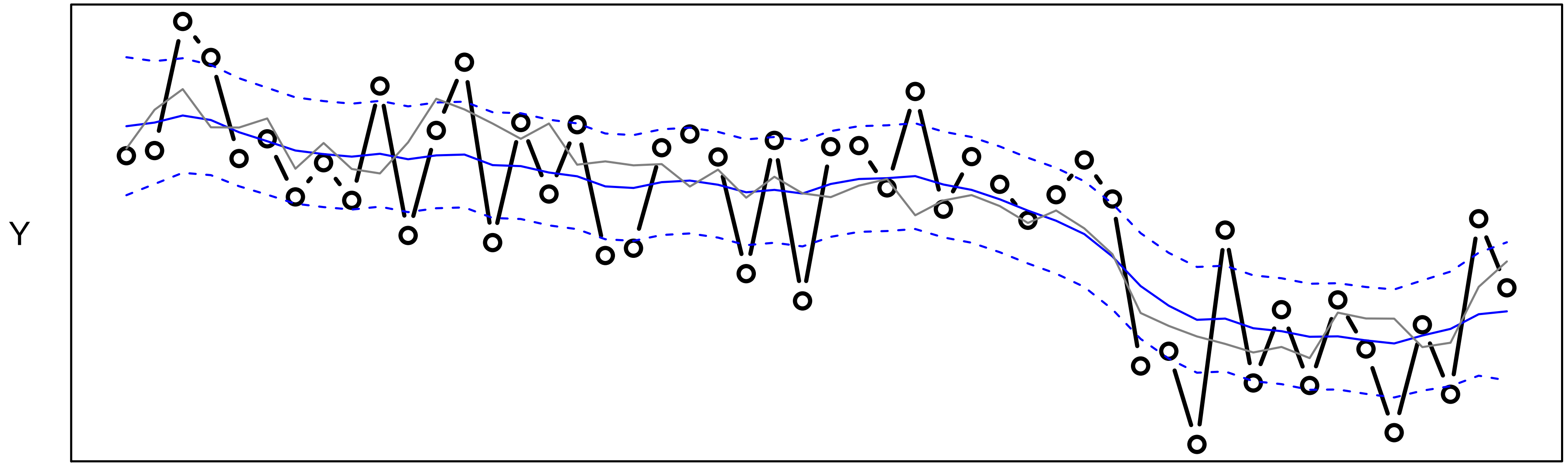
- Notice λ vector unobserved.
- Here we wish to estimate both λ and the model parameters ($\lambda_\circ, \sigma_\lambda$, and σ_ε)

Basic Kalman Filter

```cpp
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y);
  PARAMETER(logSdRw);
  PARAMETER(logSdObs);
  PARAMETER(lam0);
  int N=y.size();
  vector<Type> lam(N), lamVar(N), lamPred(N),
    lamPredVar(N), lamSmooth(N), lamSmoothVar(N),
    yPredVar(N), w(N);
  Type varLam, varY, varFrac;
  Type ans=Type(0.0);
  varLam=exp(2.0*logSdRw);
  varY=exp(2.0*logSdObs);
  lamPred(0)=lam0;
  lamPredVar(0)=varLam+Type(1000.0);
  yPredVar(0)=lamPredVar(0)+varY;
  w(0)=y(0)-lamPred(0);
  lam(0)=lamPred(0)+lamPredVar(0)/yPredVar(0)*w(0);
  lamVar(0)=lamPredVar(0)-lamPredVar(0)/yPredVar(0)*lamPredVar(0);
  ans+=Type(0.5)*log(yPredVar(0))+Type(0.5)*w(0)/yPredVar(0)*w(0);
  for(int i=1; i<N; ++i){
    lamPred(i)=lam(i-1);
    lamPredVar(i)=lamVar(i-1)+varLam;
    yPredVar(i)=lamPredVar(i)+varY;
    w(i)=y(i)-lamPred(i);
    lam(i)=lamPred(i)+lamPredVar(i)/yPredVar(i)*w(i);
    lamVar(i)=lamPredVar(i)-lamPredVar(i)/yPredVar(i)*lamPredVar(i);
    ans+=0.5*log(yPredVar(i))+0.5*w(i)/yPredVar(i)*w(i);
  }
  // ----------------- smoothing part
  lamSmooth(N-1)=lam(N-1);
  lamSmoothVar(N-1)=lamVar(N-1);
  for(int i=N-2; i>=0; --i){
    varFrac=lamVar(i)/lamPredVar(i+1);
    lamSmooth(i)=lam(i)+varFrac*(lamSmooth(i+1)-lamPred(i+1));
    lamSmoothVar(i)=lamVar(i)+
                varFrac*(lamSmoothVar(i+1)-lamPredVar(i+1))*varFrac;
  }
  REPORT(lamSmooth)
  REPORT(lamSmoothVar)
  return ans;
}
```

```r
library(TMB)
compile("kf.cpp")
dyn.load("kf.so")
data <- list(y=scan("kf.dat"))
parameters <- list(
  logSdRw=0,
  logSdObs=0,
  lam0=0
  )
newtonOption(smartsearch=FALSE)
obj<-MakeADFun(data,parameters,DLL="kf")

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

rep<-obj$report()

save(rep,file="kf.RData")
```

Estimates from the Kalman Filter



- Fast for linear and Gaussian state-space models
- Must be extended to deal with non-linear models
- Non-Gaussian models are difficult and need either simulation based filters or discrete approximation of state-space (only possible in low dimension)

But we can construct the joint likelihood

- The joint likelihood contributions from the random effects:

  $(\lambda_1 - \lambda_\circ) \sim N(0, \sigma_\lambda^2)$

  $(\lambda_2 - \lambda_1) \sim N(0, \sigma_\lambda^2)$

  ...

  $(\lambda_{50} - \lambda_{49}) \sim N(0, \sigma_\lambda^2)$

- The joint likelihood contributions from the observations:

  $y_1 \sim N(\lambda_1, \sigma^2)$

  $y_2 \sim N(\lambda_2, \sigma^2)$

  ...

  $y_{50} \sim N(\lambda_{50}, \sigma^2)$

- So the joint negative log likelihood becomes:

$$-\sum_{i=1}^{50} \log(\varphi_{\circ,\sigma_\lambda}(\lambda_i - \lambda_{i-1})) - \sum_{i=1}^{50} \log(\varphi_{\lambda_i,\sigma}(y_i))$$

We can solve this by Laplace approximation in TMB

- Code up the joint negative log likelihood
  - Declare the unobserved as `PARAMETER_VECTOR`
  - Code as if the unobserved were observed
- From inside **R** identify which quantities are to be considered as random effects (unobserved), as e.g:

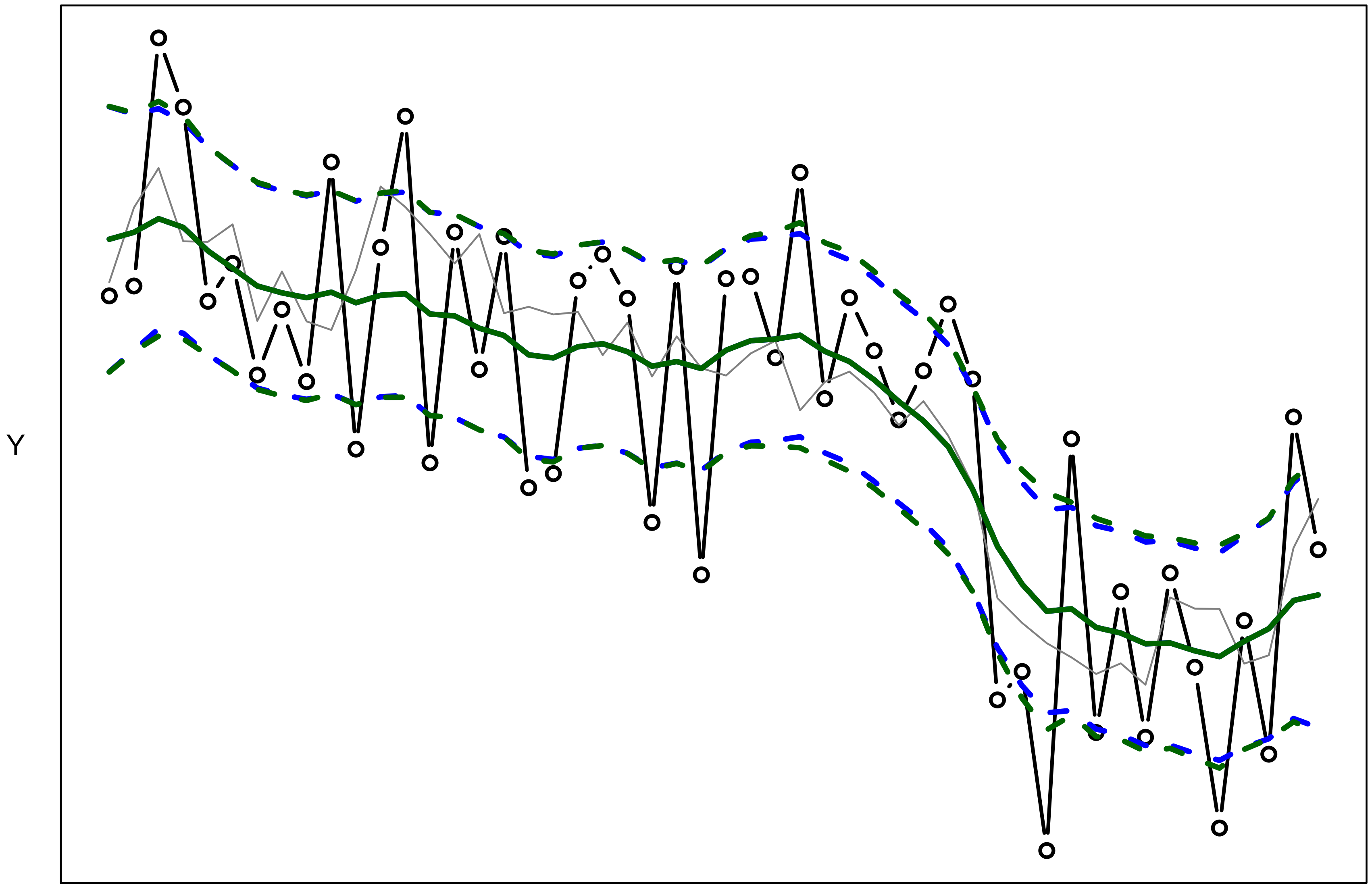
```
obj <- MakeADFun(data,parameters,random="lam",DLL="rw")
```

Random walk + noise in TMB

```cpp
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y);
  PARAMETER(logSdRw);
  PARAMETER(logSdObs);
  PARAMETER(lam0);
  PARAMETER_VECTOR(lam);

  int timeSteps=y.size();
  Type sdRw=exp(logSdRw);
  Type sdObs=exp(logSdObs);

  Type ans=-dnorm(lam(0),lam0,sdRw,true);

  for(int i=1;i<timeSteps;i++){
    ans+=-dnorm(lam(i),lam(i-1),sdRw,true);
  }

  for(int i=0;i<timeSteps;i++){
    ans+=-dnorm(y(i),lam(i),sdObs,true);
  }

  return ans;
}
```

```r
library(TMB)
compile("rw.cpp")
dyn.load(dynlib("rw"))
data <- list(y=scan("rw.dat"))
parameters <- list(
  logSdRw=0,
  logSdObs=0,
  lam0=0,
  lam=rep(0,length(data$y))
  )

obj <- MakeADFun(data,parameters,random="lam",DLL="rw")

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
save(pl,plsd,file="rw.RData")
```

Did we get the same answer?



Exercise 1: RW + noise with missing observations

a) Consider again the simple random walk plus noise example. In the file: `rwmissing.dat` six of the observations are missing (coded as 'NA'). Consider and implement changes in the program to deal with that.

b) Make a plot of the estimated process and its confidence interval (similar to the one on the previous slide) to illustrate the effect of the missing observations.

Hint: The following code is a function to help identify the observations which are not assigned (NA).

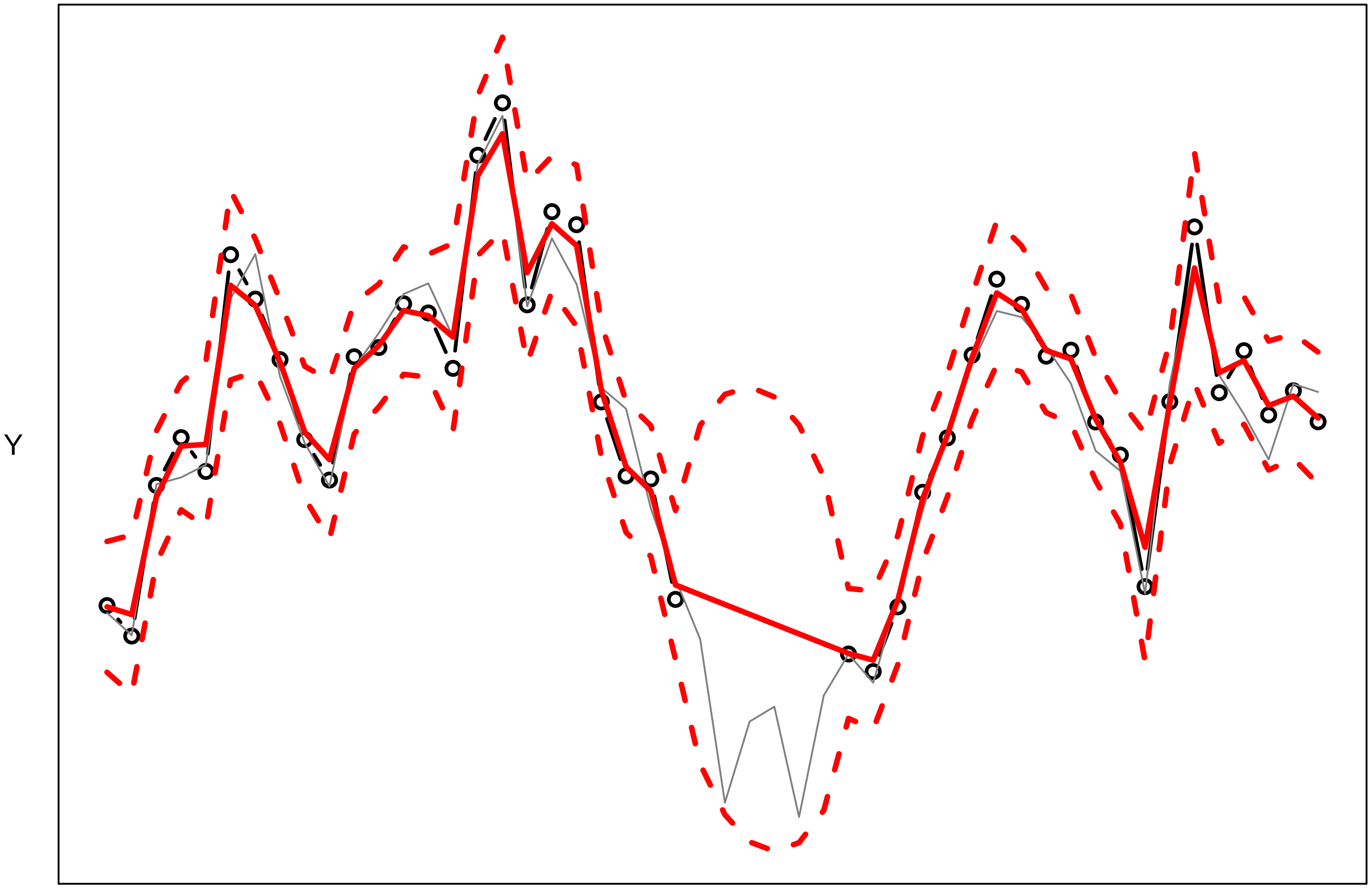
```
template<class Type>
bool isNA(Type x){
  return R_IsNA(asDouble(x));
}
```

Solution 1: RW + noise with missing observations

```
#include <TMB.hpp>

template<class Type>
bool isNA(Type x){
  return R_IsNA(asDouble(x));
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y);
  PARAMETER(logSdRw);
  PARAMETER(logSdObs);
  PARAMETER(lam0);
  PARAMETER_VECTOR(lam);

  int timeSteps=y.size();
  Type sdRw=exp(logSdRw);
  Type sdObs=exp(logSdObs);

  Type ans=-dnorm(lam(0),lam0,sdRw,true);

  for(int i=1;i<timeSteps;i++){
    ans+=-dnorm(lam(i),lam(i-1),sdRw,true);
  }

  for(int i=0;i<timeSteps;i++){
    if(!isNA(y(i))){
      ans+=-dnorm(y(i),lam(i),sdObs,true);
    }
  }

  return ans;
}
```

```
library(TMB)
compile("rwmissing.cpp")
dyn.load("rwmissing.so")
data <- list(y=scan("rwmissing.dat"))
parameters <- list(
  logSdRw=0,
  logSdObs=0,
  lam0=0,
  lam=rep(0,length(data$y))
  )

obj <- MakeADFun(data,parameters,random="lam",DLL="rwmi

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr, "Est")
plsd <- as.list(sdr, "Std")
```

Solution 1 (cont.)



That’s basically it(!)

- Seems like a too simple example
- This approach can be useful very generally, e.g:
- Higher dimension
- Some non-linearity
- Some non-normality in the process and observations
- So let’s look at a few examples...

Higher dimension

- Consider the model:
  - The unobserved part:

$$\alpha_i^{(1)} = \alpha_{i-1}^{(1)} + \varepsilon_i^{(1)}$$
$$\alpha_i^{(2)} = \alpha_{i-1}^{(2)} + \varepsilon_i^{(2)}$$
$$\alpha_i^{(3)} = \alpha_{i-1}^{(3)} + \varepsilon_i^{(3)}$$

  - The observed part:

$$y_i^{(1)} = \alpha_i^{(1)} + \eta_i^{(1)}$$
$$y_i^{(2)} = \alpha_i^{(2)} + \eta_i^{(2)}$$
$$y_i^{(3)} = \alpha_i^{(3)} + \eta_i^{(3)}$$

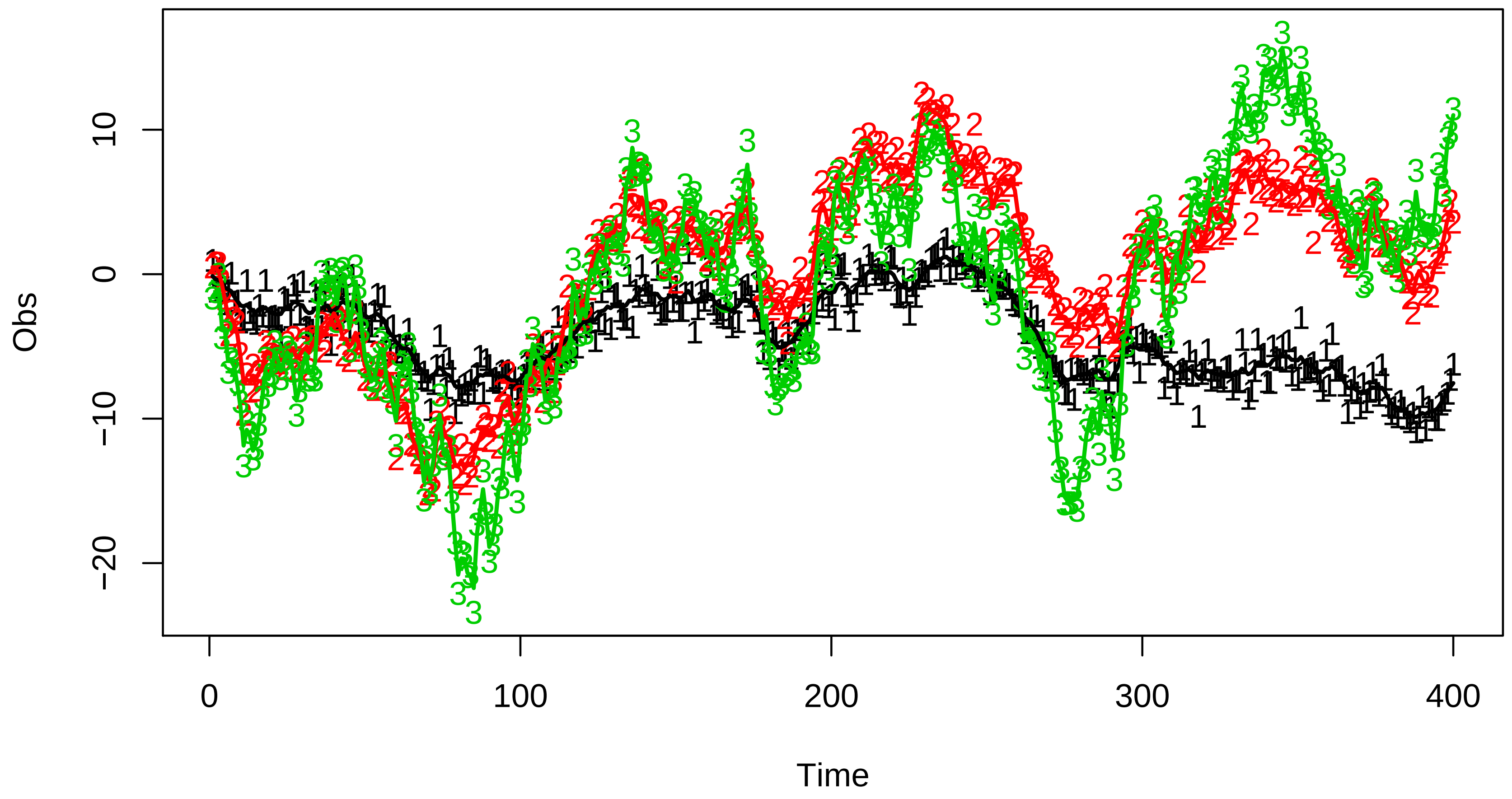
  - Let's start by all noise terms have separate variance and are independent.

Multivariate RW

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_ARRAY(obs); /* stateDim x timeSteps */
  PARAMETER_VECTOR(logsds);
  PARAMETER_VECTOR(logsdObs);
  PARAMETER_ARRAY(u); /* State matrix */
  int stateDim=obs.dim(0);
  int timeSteps=obs.dim(1);
  vector<Type> sds=exp(logsds);
  vector<Type> sdObs=exp(logsdObs);

  matrix<Type> cov(stateDim,stateDim);
  cov.setZero();
  for(int i=0;i<stateDim;i++)
    for(int j=0;j<stateDim;j++)
      if(i==j)cov(i,j)=sds(i)*sds(j);
  using namespace density;
  MVNORM_t<Type> neg_log_density(cov);

  /* Define likelihood */
  Type ans=0;
  for(int i=1;i<timeSteps;i++)
    ans+=neg_log_density(u.col(i)-u.col(i-1));
  for(int i=0;i<timeSteps;i++)
    ans-=dnorm(vector<Type>(obs.col(i)),
               vector<Type>(u.col(i)),
               sdObs,true).sum();
  return ans;
}
```

```
library(TMB)
compile("mvrw.cpp")
dyn.load(dynlib("mvrw"))
obs<-read.table("mvrw.dat", header=FALSE)
data <- list(obs=t(obs))
param <- list(
  logsds=rep(0,dim(obs)[2]),
  logsdObs=rep(0,dim(obs)[2]),
  u=data$obs*0
  )
obj <- MakeADFun(data,param,random="u",DLL="mvrw")

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
```

Estimates from independent multivariate RW



Exercise 2: Correlation between the processes

- In the multivariate random walk example the three different α processes were assumed independent. If $\alpha^{(1)}$ and $\alpha^{(2)}$ are two environmental indices and $\alpha^{(3)}$ is abundance of our most economically important fish stock, then we may be interested in estimating how correlated they are. Consider the model:
- Define $\Delta\alpha_t = \begin{pmatrix} \alpha_t^{(1)} - \alpha_{t-1}^{(1)} \\ \alpha_t^{(2)} - \alpha_{t-1}^{(2)} \\ \alpha_t^{(3)} - \alpha_{t-1}^{(3)} \end{pmatrix}$
- Let's assume $\Delta\alpha_t \sim N_{3\times 3}(0, \Sigma)$ where: $\Sigma_{i,j} = \rho^{|i-j|}\sigma_i\sigma_j$
- Here ρ is a new model parameter, which should be bounded between -1 and 1.

Hint: The following code is a function which could be helpful for bounding within (-1,1).

```
/* Parameter transform */
template <class Type>
Type bound(Type x){
  return Type(2)/(Type(1) + exp(-x)) - Type(1);
}
```

Solution 2: Multivariate RW with correlation

```cpp
#include <TMB.hpp>
template <class Type>
Type bound(Type x){
  return Type(2)/(Type(1) + exp(-x)) - Type(1);
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_ARRAY(obs); /* stateDim x timeSteps */
  PARAMETER(transf_rho);
  PARAMETER_VECTOR(logsds);
  PARAMETER_VECTOR(logsdObs);
  PARAMETER_ARRAY(u); /* State */
  int timeSteps=obs.dim(1);
  int stateDim=obs.dim(0);
  Type rho=bound(transf_rho);
  vector<Type> sds=exp(logsds);
  vector<Type> sdObs=exp(logsdObs);

  matrix<Type> cov(stateDim,stateDim);   cov.setZero();
  for(int i=0;i<stateDim;i++)
    for(int j=0;j<stateDim;j++){
      if(i==j){
        cov(i,j)=sds(i)*sds(j);
      }else{
        cov(i,j)=pow(rho,Type(abs(i-j)))*sds(i)*sds(j);
      }
    }
  density::MVNORM_t<Type> neg_log_density(cov);
  Type ans=0;
  for(int i=1;i<timeSteps;i++)
    ans+=neg_log_density(u.col(i)-u.col(i-1));
  for(int i=0;i<timeSteps;i++)
    ans-=dnorm(vector<Type>(obs.col(i)),
               vector<Type>(u.col(i)),sdObs,1).sum();
  ADREPORT(rho); REPORT(cov);
  return ans;
}
```

```r
library(TMB)
compile("mvrw.cpp")
dyn.load(dynlib("mvrw"))
obs<-read.table("mvrw.dat", header=FALSE)
data <- list(obs=t(obs))
param <- list(
  transf_rho=0.1,
  logsds=rep(0,dim(obs)[2]),
  logsdObs=rep(0,dim(obs)[2]),
  u=data$obs*0
  )

obj <- MakeADFun(data,param,random="u",DLL="mvrw")
obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
```

```
> c(sdr$value, sdr$sd)

       rho
0.89831761 0.01959993
```

Non-linear and multivariate state-space model

- The unobserved part:

$$\lambda_i^{(1)} = \lambda_{i-1}^{(1)} + \varepsilon_i^{(1)}$$
$$\lambda_i^{(2)} = \phi\lambda_{i-1}^{(2)} + \varepsilon_i^{(2)}$$

- The observed part:

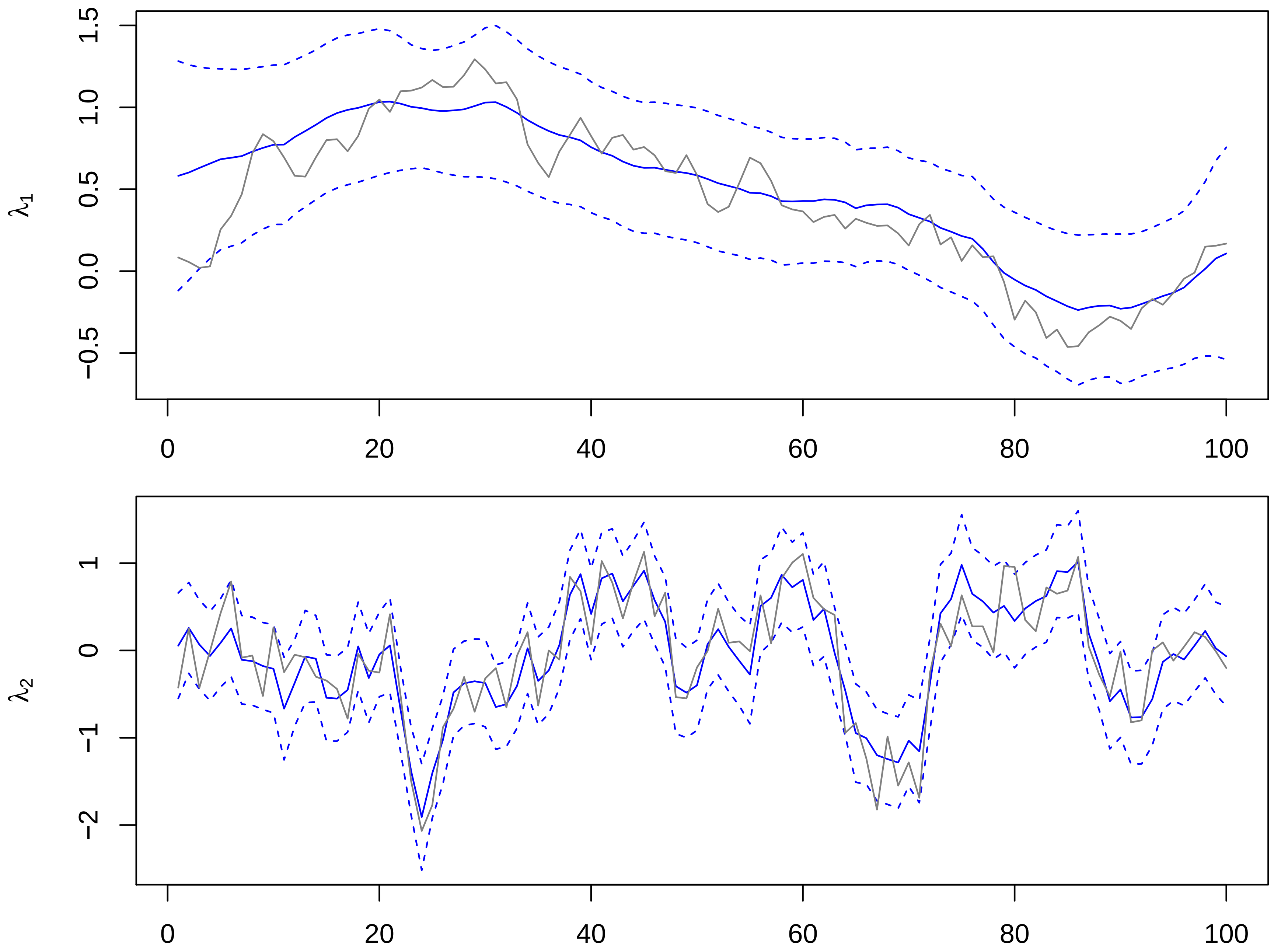
$$y_i^{(1)} = e^{\lambda_i^{(1)}}/(1 + e^{\lambda_i^{(1)}}) + \eta_i^{(1)}$$
$$y_i^{(2)} = \lambda_i^{(2)} + \eta_i^{(2)}$$
$$y_i^{(3)} = \lambda_i^{(1)}\lambda_i^{(2)} + \eta_i^{(3)}$$

- All noise terms have separate variance and are independent.

Non-linear and multivariate state-space model

```cpp
#include <TMB.hpp>
template <class Type>
Type bound(Type x){
  return Type(2)/(Type(1) + exp(-Type(2) * x)) - Type(1);
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_ARRAY(obs); /* stateDim x timeSteps */
  PARAMETER(trans_phi)
  PARAMETER_VECTOR(logsds);
  PARAMETER_VECTOR(logsdObs);
  PARAMETER_ARRAY(u); /* State matrix */
  int stateDim=u.dim(0);
  int obsDim=obs.dim(0);
  int timeSteps=u.dim(1);
  vector<Type> sds=exp(logsds);
  vector<Type> sdObs=exp(logsdObs);
  Type phi=bound(trans_phi);

  vector<Type> predU(stateDim);
  vector<Type> predObs(obsDim);

  /* Define likelihood */
  Type ans=0;
  for(int i=1;i<timeSteps;i++){
    predU(0)=u(0,i-1);
    predU(1)=phi*u(1,i-1);
    ans-=dnorm(vector<Type>(u.col(i)),predU,sds,true).sum();
  }
  for(int i=0;i<timeSteps;i++){
    predObs(0)=exp(u(0,i))/(Type(1.0)+exp(u(0,i)));
    predObs(1)=u(1,i);
    predObs(2)=u(0,i)*u(1,i);
    ans-=dnorm(vector<Type>(obs.col(i)),
               predObs,sdObs,true).sum();
  }
  return ans;
}
```

```r
library(TMB)
compile("nl.cpp")
dyn.load(dynlib("nl"))
obs<-read.table("nonlin.dat", header=FALSE)
data <- list(obs=t(obs))
param <- list(
  trans_phi=0,
  logsds=rep(0,2),
  logsdObs=rep(0,dim(obs)[2]),
  u=matrix(0,nrow=2, ncol=dim(obs)[1])
  )

obj <- MakeADFun(data,param,random="u",DLL="nl")

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
```



λ_1
−0.5
0.0
0.5
1.0
1.5
0
20
40
60
80
100
λ_2
−2
−1
0
1
0
20
40
60
80
100

Discrete valued time series

- Examples from Kuk & Cheng (1999).
- The model:

$$y_i \sim \text{Pois}(\lambda_i) \ , \ \text{where}$$
$$\log(\lambda_i) = X_i b + u_i \ , \ \text{and}$$
$$u_i = a u_{i-1} + \varepsilon_i$$

Here, X_i is a design matrix of covariates, b is a vector of regression parameters and u_i is an AR(1). The dimension of b is 6 and i=1,...,168.

	β_1	β_2	β_3	β_4	β_5	β_6	a	σ
TMB	0.242	-3.814	0.162	-0.482	0.413	-0.011	0.627	0.538
ADMB-RE	0.242	-3.814	0.162	-0.482	0.413	-0.011	0.627	0.538
Std. dev.	0.270	2.76	0.15	0.16	0.13	0.13	0.19	0.15
Kuk & Cheng	0.244	-3.82	0.162	-0.478	0.413	-0.0109	0.665	0.519

Code: Discrete valued time series

```cpp
#include <TMB.hpp>
template <class Type>
Type bound(Type x){
  return Type(2)/(Type(1) + exp(-Type(2) * x)) - Type(1);
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y)
  DATA_MATRIX(X);
  PARAMETER_VECTOR(b);
  PARAMETER(trans_a);
  PARAMETER(logSigma);
  PARAMETER_VECTOR(u);
  int timeSteps=y.size();
  Type a=bound(trans_a);
  Type sigma=exp(logSigma);

  /* Define likelihood */
  Type ans=0;

  ans-=dnorm(u(0),Type(0),sigma/sqrt(Type(1)-a*a),true);

  for(int i=1;i<timeSteps;i++)
    ans-=dnorm(u(i),a*u(i-1),sigma,true);

  vector<Type> eta = X*b;
  Type lambda;

  for(int i=0;i<timeSteps;i++){
    lambda=exp(eta(i)+u(i));
    ans-=dpois(y(i),lambda,true);
  }
  return ans;
}
```

```r
library(TMB)
compile("pol.cpp")
dyn.load(dynlib("pol"))
dat<-read.table("pol.dat", header=FALSE)
data <- list(y=dat[,1], X=as.matrix(dat[,-1]))
param <- list(
  b=rep(0,ncol(data$X)),
  trans_a=0,
  logSigma=0,
  u=rep(0,length(data$y))
  )

obj <- MakeADFun(data,param,random="u",DLL="pol")

obj$fn()
obj$gr()
opt<-nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
```

Exercise 3: A theta logistic population model

A theta logistic population model is defined for the log-transformed population size as a nonlinear function of its previous size in the following way:

$$\begin{aligned} X_t &= X_{t-1} + r_0 \left(1 - \left(\frac{\exp(X_{t-1})}{K}\right)^{\theta}\right) + e_t, \\ Y_t &= X_t + u_t, \end{aligned}$$

where $e_t \sim N(0, Q)$ and $u_t \sim N(0, R)$.

Data for this model is available in the file: `theta.dat`

a) Fit the model to data and estimate the five model parameters.

Hint: It is helpful to log transform the parameters and initialize $\log(K)$ to around 6.

Solution 3: A theta logistic population model

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  /* Data section */
  DATA_VECTOR(Y);
  DATA_VECTOR_INDICATOR (keep , Y);
  /* Parameter section */
  PARAMETER_VECTOR(X);
  PARAMETER(logr0);
  PARAMETER(logtheta);
  PARAMETER(logK);
  PARAMETER(logQ);
  PARAMETER(logR);
  /* Procedure section */
  Type r0=exp(logr0);
  Type theta=exp(logtheta);
  Type K=exp(logK);
  Type Q=exp(logQ);
  Type R=exp(logR);
  int timeSteps=Y.size();
  Type ans=0;
  Type m;
  for(int i=1;i<timeSteps;i++){
    m=X[i-1]+r0*(1.0-pow(exp(X[i-1])/K,theta));
    ans-=dnorm(X[i],m,sqrt(Q),true);
  }
  for(int i=0;i<timeSteps;i++){
    ans-=keep(i)*dnorm(Y[i],X[i],sqrt(R),true);
  }
  return ans;
}
```

```
Y<-scan('theta.dat', quiet=TRUE)

library(TMB)
compile("theta.cpp")
dyn.load(dynlib("theta"))
data <- list(Y=Y)
param <- list(
  X=data$Y*0,
  logr0=0,
  logtheta=0,
  logK=6,
  logQ=0,
  logR=0
  )

obj <- MakeADFun(data,param,random="X",DLL="theta")
obj$fn()
obj$gr()
opt <- nlminb(obj$par,obj$fn,obj$gr)

sdr<-sdreport(obj)
pl <- as.list(sdr,"Est")
plsd <- as.list(sdr,"Std")
```

Summary

- Laplace approximation is a very general and flexible way of dealing with state-space models
- TMB is very efficient.
- Besides state-space models the same technique is useful for a lot of other general random effects models.
- Growing set of examples at `http://tmb-project.org`
- For spatial models and high dimensional state-space models it is useful to look at more optimized ways to specify the distribution of the random effects (GMRF, AR, ...) available in TMB.
- Laplace approximation not always appropriate e.g. if you have discrete random effects.

State-space assessment

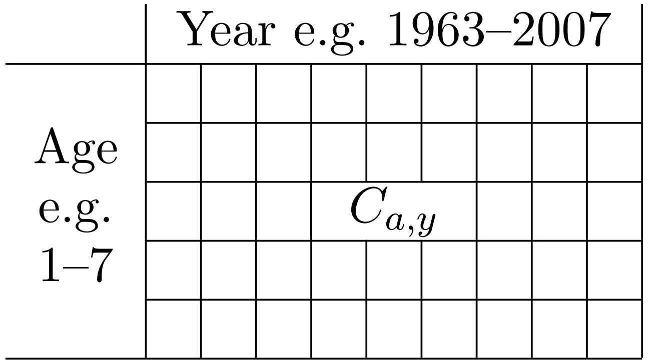
Fish Stock Assessment

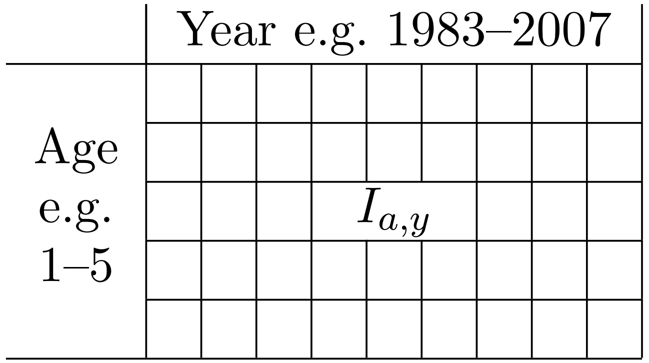
Problem: How many fish (relative or absolute) are left in the ocean?

Data:

$C_{a,y}$**:** Yearly catches (divided into age–classes)

$I_{a,y}$**:** Scientific surveys



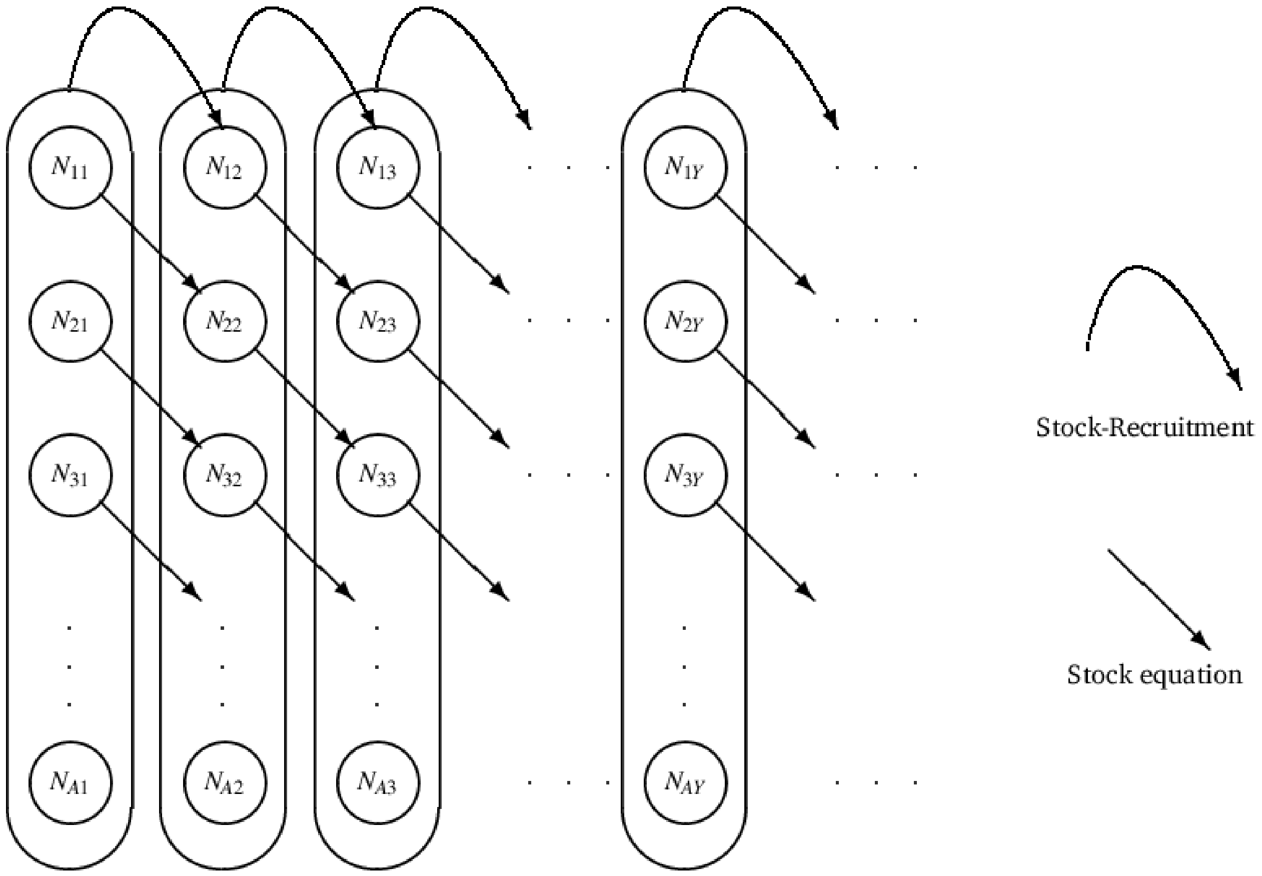


Often we have catches ($C_{f,a,y}$) from more than one fleet and indices ($I_{s,a,y}$) from more than one survey, but here we keep it simple.

Important output: "Reference points" SSB, $\overline{F}$, ...

Assessment Models

- Based on a standard set of equations describing the structure of the system
- Can include selectivity
- Can include different assumptions about 'natural' mortality
- Can include different effort numbers
- Can include different assumptions about recruitment
- When all of these things have been taken into account:



The number of fish is assumed to be proportional to the catch

Deterministic models

- A deterministic model is a model where observation noise is ignored
- Typically catches are assumed known without error
- Most commonly applied fish stock assessment models are (semi-)deterministic
- These algorithms work (very simply put) by:

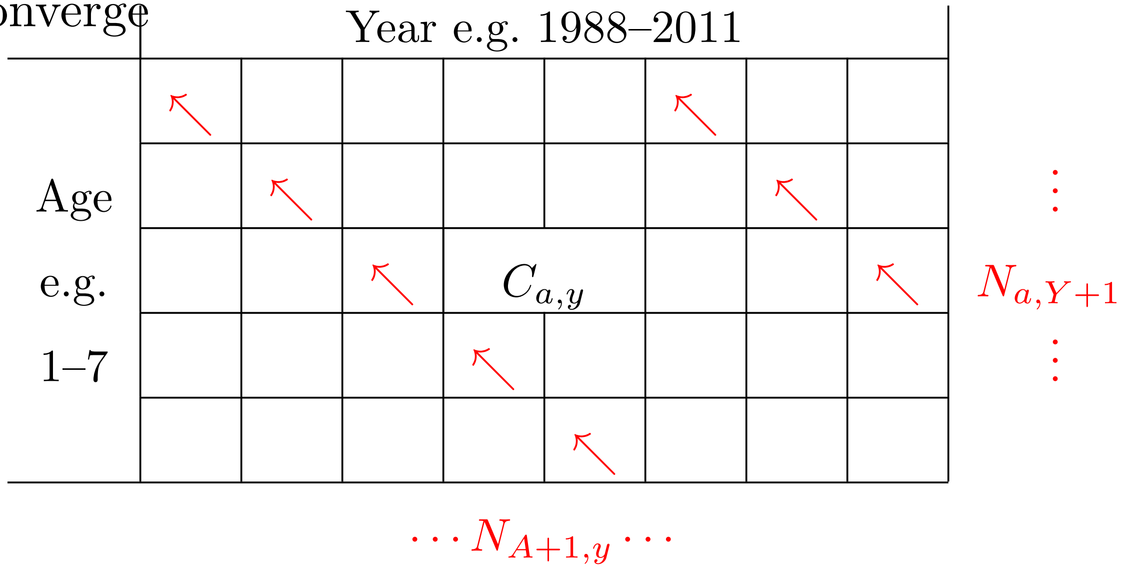
  0: Guess the number of survivors $N_{A+1,y}$ and $N_{a,Y+1}$

  1: Back calculate ($\nwarrow$) all $N_{a,y}$ by subtracting catch and natural mortality

  2: Use surveys to adjust all $N_{a,y}$ and update survivors accordingly

  3: Repeat 1-3 until survivors converge

- Doing 0-1 just ones is known as Virtual Population Analysis



Features of deterministic models

+ Super fast to compute

+ Fairly simple to explain the path from data to stock numbers (especially VPA)

– Difficult to explain why it works (converges), and what a solution mean

– These algorithms contain many ad-hoc settings (e.g. shrinkage, tapered time weights) that makes them less objective

– No quantification of uncertainties within model

? What exactly is the model

  - The assumptions are difficult to identify and verify
  - With no clearly defined model more ad-hoc methods are needed to make predictions

– No framework for comparing models (different settings)

A full parametric statistical model

- The log catches are assumed to follow:

$$\log(C_{a,y}) \sim \mathcal{N}\left(\log\left(\frac{F_{a,y}}{Z_{a,y}}(1-e^{-Z_{a,y}})N_{a,y}\right), \sigma_c^2\right) \ \ , \text{ where}$$

$$F_{a,y} = f_y S_a, \text{ with } S_{a=5} = S_{a=6} = S_{a=7} = 1, \text{ and } Z_{a,y} = F_{a,y} + M_{a,y}$$

- The log catches from the survey are assumed to follow:

$$\log(I_{a,y}) \sim \mathcal{N}\left(\log\left(Q_a e^{-Z_{a,y}T} N_{a,y}\right), \sigma_s^2\right) \ \ , \text{ where}$$

  T is the fraction into the year where the survey is taken, and Q_a is catchability parameter.

- The stock sizes are assumed to follow:

$$N_{a,y} = N_{a-1,y-1} e^{-Za-1,y-1}$$

  Notice that it does not define N in the first year and for the youngest age.

- So the model parameters are the undefined N's, f_y, S_a, Q_a, σ_c, and σ_s

Features of full parametric statistical models

+ Acknowledges observation noise
+ All model assumptions are transparent
+ Different model assumptions can be tested against each other (e.g. is $F_5 = F_6$?)
+ Different data sources can be included and correctly and objectively weighted
+ Estimation of uncertainties are an integrated part of the model

− Trade-off between the number model parameters and flexibility of the model (e.g. $F_{a,y}$ vs. $F_{a,y} = S_a f_y$)

− Too often ad-hoc solutions are needed (e.g. fixing variance parameters, or setting fixed penalties)

− More advanced software needed (ADMB or TMB)

State-space assessment models

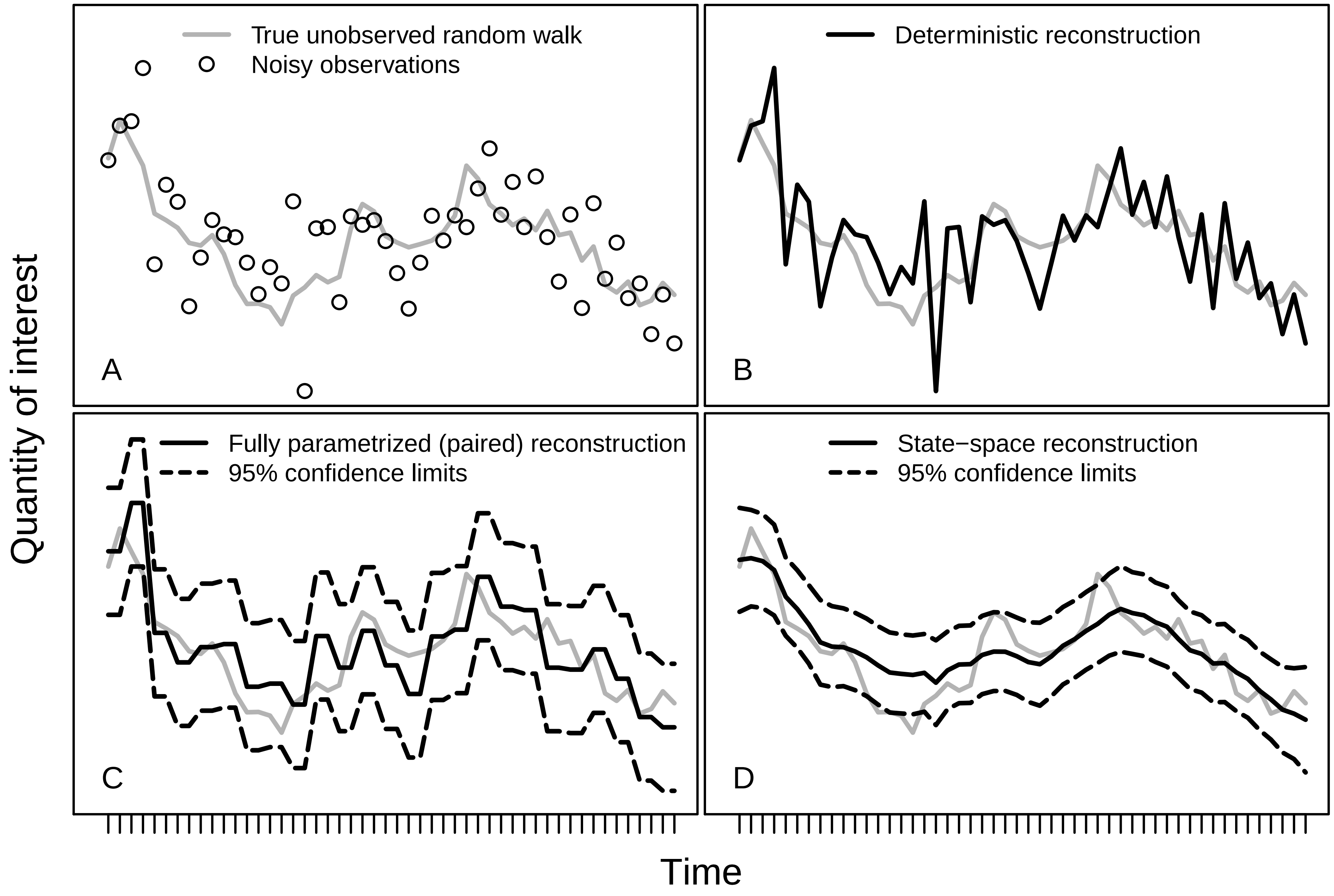
- This model class[a] is used in most other quantitative fields
- It is a very useful extension to full parametric statistical models.
- Introduced for stock assessment by Gudmundsson (1987,1994) and Fryer (2001).
- The reason state-space models have not been more frequently used in stock assessment is that software to easily handle these models has not been available
- Has recently received increased attention (e.g. Brinch et al., 2011; Gudmundsson and Gunnlaugsson, 2012; Berg et al., 2013)
- Can give very flexible models with low number of model parameters
- For instance we can include things like:

$$F_{3,y} \text{ is a random walk with yearly variance } \sigma^2$$

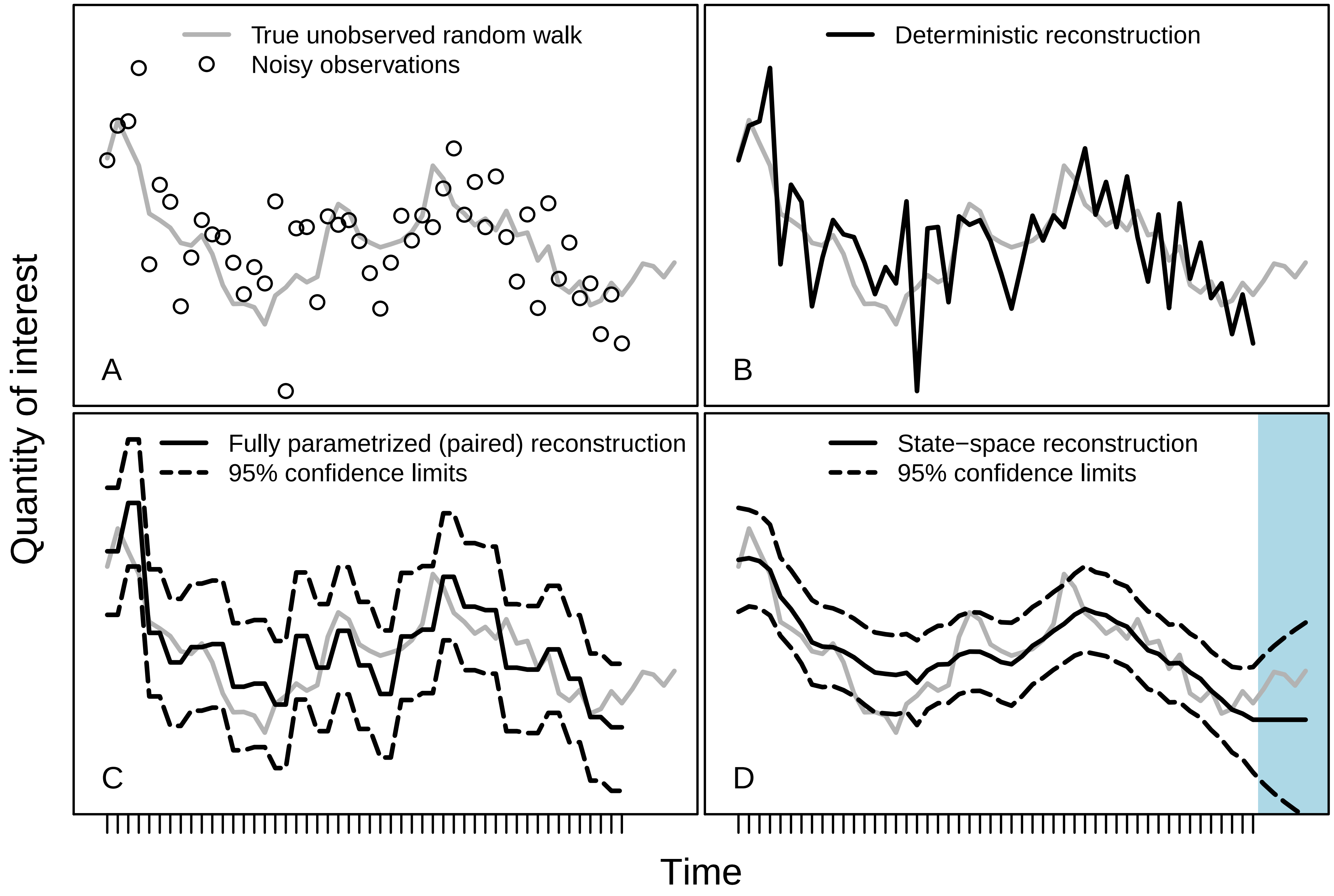
- Importantly σ^2 is a model parameter estimated in the model.

[a] a.k.a. random effects models, mixed models, latent variable models, hierarchical models, ...

Illustration of the three types of models

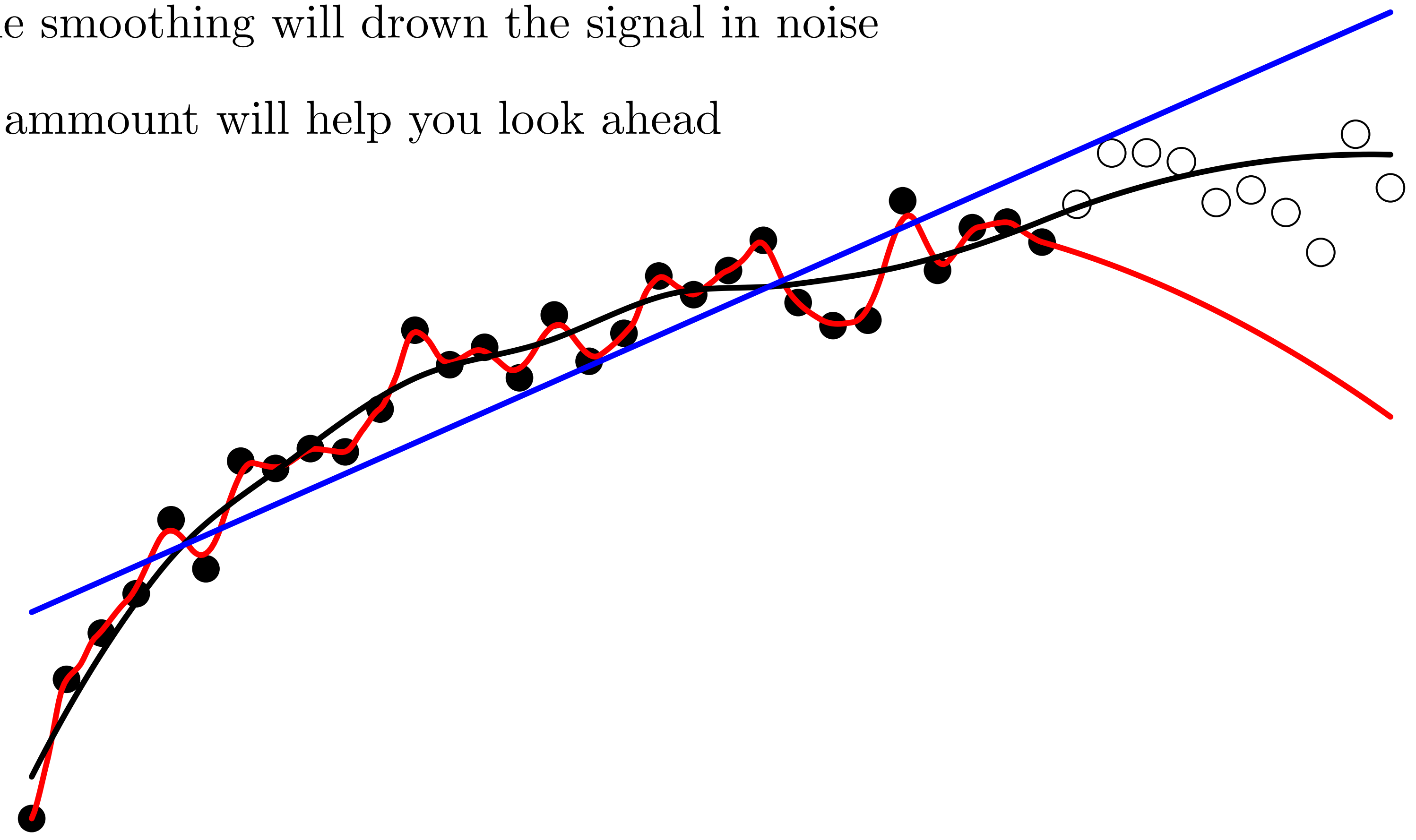


Only one model type can predict



Is smoothing evil?

- Too much smoothing will bias the signal
- Too little smoothing will drown the signal in noise
- Correct ammount will help you look ahead



- Correct amount should not be subjective.

Model

States are the random variables that we don't observe ($N_{a,y}$, $F_{a,y}$)

$$\begin{pmatrix} \log(N_y) \\ \log(F_y) \end{pmatrix} = T \begin{pmatrix} \log(N_{y-1}) \\ \log(F_{y-1}) \end{pmatrix} + \eta_y$$

Observations are the random variables that we do observe ($C_{a,y}$, $I_{a,y}^{(s)}$)

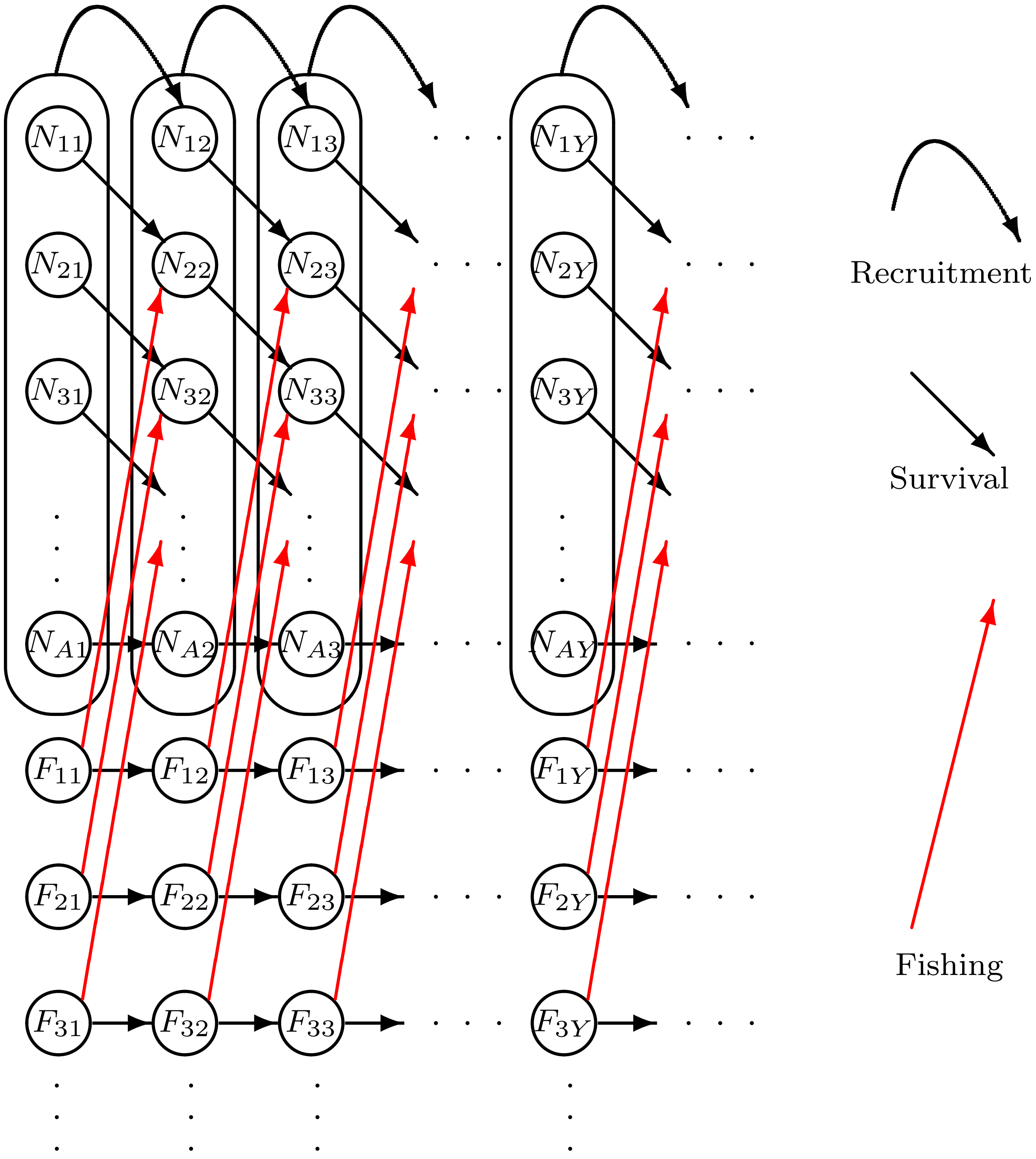
$$\begin{pmatrix} \log(C_y) \\ \log(I_y^{(s)}) \end{pmatrix} = O \begin{pmatrix} N_y \\ F_y \end{pmatrix} + \varepsilon_y$$

Model and parameters are what describes the distribution of states and observations through T, O, η_y, and ε_y.

Parameters: Survey catchabilities, S-R parameters, process and observation variances.

All model equation are as expected:

- Standard stock equation
- Standard stock recruitment (B-H, Ricker, or RW)
- Standard equations for total landings and survey indices


N_{11}
N_{12}
N_{13}
N_{1Y}
N_{21}
N_{22}
N_{23}
N_{2Y}
N_{31}
N_{32}
N_{33}
N_{3Y}
N_{A1}
N_{A2}
N_{A3}
N_{AY}
F_{11}
F_{12}
F_{13}
F_{1Y}
F_{21}
F_{22}
F_{23}
F_{2Y}
F_{31}
F_{32}
F_{33}
F_{3Y}
Recruitment
Survival
Fishing

Exercise: Laplace approximation accuracy

- The negative binomial model:

$$X_i \sim \mathrm{NB}(N, p) \quad , \quad i = 1, \ldots, n$$

  is interesting because we know that it is identical to the model

$$X_i \sim \mathrm{Pois}(\lambda_i) \quad , \quad \text{where } \lambda_i \sim \Gamma(N, \tfrac{1-p}{p}) \quad , \quad i = 1, \ldots, n$$

  where λ's are random effects

- Implement the model both ways and compare the estimates. The following data can be used:

```
set.seed(123)
data <- list(y=rnbinom(1000,10,.7))
```

- We can also formulate the model so the random effects are normal and then transform to a gamma distribution. The details are:

$$X_i \sim \mathrm{Pois}(\lambda_i) \quad , \quad \text{where } \lambda_i = G^{-1}_{N,\frac{1-p}{p}}(\Phi(z_i)) \quad , \quad z_i \sim N(0,1) \quad , \quad i = 1, \ldots, n$$

- here Φ is the cdf for the standard normal (`pnorm` in TMB) and $G^{-1}_{a,b}$ is the inverse cdf of the Γ distribution with parameters a and b (`qgamma` in TMB)

- Implement the model this to see if that improves the accuracy of the Laplace approximation

Exercise (extra, if time): Laplace in R

- Last time we talked about implementing a random walk+noise model (examples are in the files `rw.R`, `rw.cpp`, and `rw.RData`)
- But we now understand the Laplace approximation well, so maybe we can do this relative simple model in plain R?
- Try implementing the exact same model in plain R and compare.
- Hint: useful functions are: `optimHess` and `det`.