Simulating and estimating a state-space random walk model in R and TMB

Christopher L. Cahill

Quantitative Fisheries Center

Michigan State University

Random walk in R and TMB

Goal:

- estimate/simulate a state-space model

Steps:

- A bit of math
- Simulate fake data in R
- Build estimation model in TMB
- Pray that it converges, weep if it does not (how to assess?)
- How to deal with missing data

Math for a simple state-space random walk

$$
\begin{aligned}
&\lambda_i = \lambda_{i-1} + \eta_i \\
&Y_i = \lambda_i + \varepsilon_i \\
&\text{where } i = 1 \ldots 43, \eta_i \sim N\left(0, \sigma^2_{rw}\right), \\
&and\; \varepsilon_i \sim N\left(0, \sigma^2_{obs}\right) \text{ all independent.}
\end{aligned}
$$

R code for simulating a random walk:

rw.R

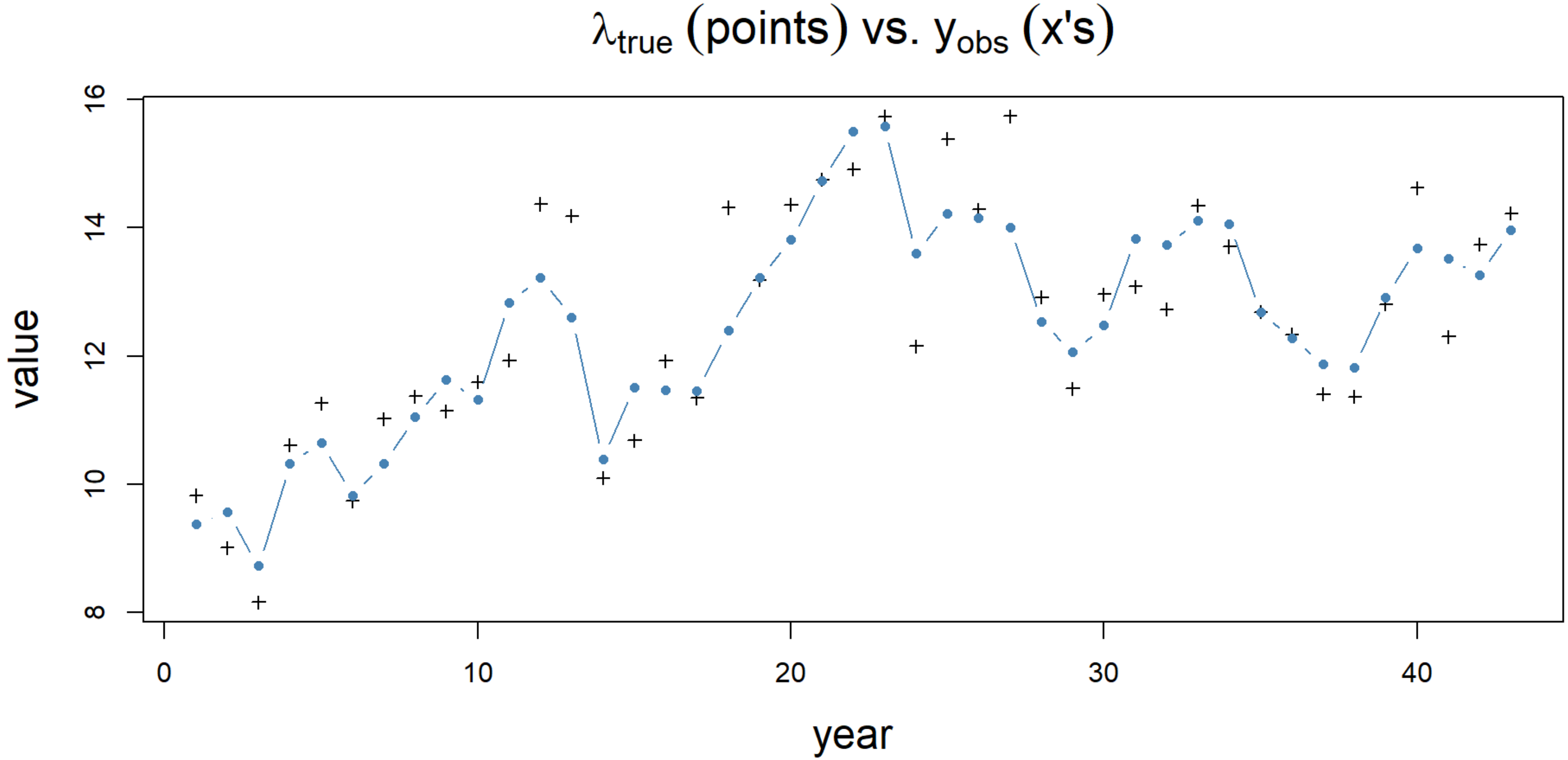
```
# set leading parameters
years = 1:43
lam0 = 10 # initial value
sd_rw = 1 # stdev of process
sd_obs = 0.8 # stdev of observations
lams = rep(NA, length(years)) # true lambdas
y_obs = rep(NA, length(years)) # observed data
set.seed(1) # ensure "random" data are same

lams[1] = rnorm(1, lam0, sd_rw) # initialize the stochastic process

# do the random walk, add in process error:
for(i in 2:length(years)){
  lams[i] = rnorm(1, lams[i-1], sd_rw)
}

# add observation error to true (latent) process:
y_obs = rnorm(length(lams), lams, sd_obs)
```

Visualizing the simulated data



TMB code for estimating a random walk:

rw.cpp

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y_obs);                                    // observed data
  PARAMETER(ln_sd_rw);                                   // log(sd) process
  PARAMETER(ln_sd_obs);                                  // log(sd) observation
  PARAMETER(lam0);                                       // initial state
  PARAMETER_VECTOR(lams);                                // random effects

  int n_year = y_obs.size();
  Type sd_rw = exp(ln_sd_rw);
  Type sd_obs = exp(ln_sd_obs);

  // random effects:
  jnll -= dnorm(lams(0),lam0,sd_rw,true);               // pr(inital state)
  for(int i=1; i<n_year; i++){
    jnll -= dnorm(lams(i),lams(i-1),sd_rw,true);         // pr(subsequent states)
  }

  // likelihood:
  for(int i=0; i<n_year; i++){
    jnll -= dnorm(y_obs(i),lams(i),sd_obs,true);         // pr(observations)
  }

  return jnll;                                           // joint neg log likelihood
}
```

Pulling it all together in `rw.R`

```r
library(TMB)
compile("rw.cpp")
```

```
[1] 0
```

```r
# create dynamically linked library:
dyn.load(dynlib("rw"))

# create a tagged data list:
data <- list(y_obs = y_obs)

# create a tagged parameter list w/ start values:
parameters <- list(ln_sd_rw = 0,
                   ln_sd_obs = 0,
                   lam0 = 0,
                   lams = rep(0, length(data$y_obs))
)

# create objective function based on template:
obj <- MakeADFun(data, parameters, random = "lams", DLL= "rw")
```

Pulling it all together cont'd

```r
obj$fn() # return the objective f(x) value
```

```
Optimizing tape... Done
iter: 1  value: 124.1655 mgc: 15.7428 ustep: 1
iter: 2  mgc: 8.881784e-15
```

```
[1] 105.1815
attr(,"logarithm")
[1] TRUE
```

Pulling it all together cont'd

```
obj$gr() # examine par gradients
```

```
iter: 1  mgc: 8.881784e-15
Matching hessian patterns... Done
outer mgc:  34.28911
```

```
[1] -34.28911 -12.98452  -5.88800
```

Run the optimization:

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

```
iter: 1  mgc: 8.881784e-15
iter: 1  mgc: 8.881784e-15
outer mgc:  34.28911
iter: 1  value: 144.2216 mgc: 1.358261 ustep: 1
iter: 2  mgc: 9.298118e-16
iter: 1  mgc: 9.298118e-16
outer mgc:  13.72374
iter: 1  value: 117.857 mgc: 3.022377 ustep: 1
iter: 2  mgc: 1.970646e-15
iter: 1  mgc: 1.970646e-15
outer mgc:  15.32134
iter: 1  value: 117.4061 mgc: 7.513776 ustep: 1
iter: 2  mgc: 3.941292e-15
iter: 1  value: 112.2716 mgc: 0.7330823 ustep: 1
iter: 2  mgc: 2.220446e-15
```

Get standard deviations

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

```
iter: 1  mgc: 8.21565e-15
outer mgc:  3.136849e-05
iter: 1  value: 82.22413 mgc: 0.003330541 ustep: 1
iter: 2  mgc: 6.439294e-15
outer mgc:  0.01252672
iter: 1  value: 82.16155 mgc: 0.003337209 ustep: 1
iter: 2  mgc: 8.715251e-15
outer mgc:  0.01259331
iter: 1  value: 82.20457 mgc: 0.003330541 ustep: 1
iter: 2  mgc: 8.65974e-15
outer mgc:  0.04996776
iter: 1  value: 82.18115 mgc: 0.003337209 ustep: 1
iter: 2  mgc: 7.438494e-15
outer mgc:  0.05006597
iter: 1  value: 82.19284 mgc: 0.002730473 ustep: 1
iter: 2  mgc: 6.661338e-15
```

Get standard deviations

```r
print(sdr)
```

```
sdreport(.) result
              Estimate Std. Error
ln_sd_rw   -0.50223741  0.3639872
ln_sd_obs   0.07582131  0.1690006
lam0        9.57015318  0.9388089
Maximum gradient component: 3.136849e-05
```

- Note: may need to apply a bias correction
- see `?sdreport`
- **Important:** are the SDs too big?

Are diagnostics consistent with convergence?

```
final_gradient = obj$gr(opt$par)
```

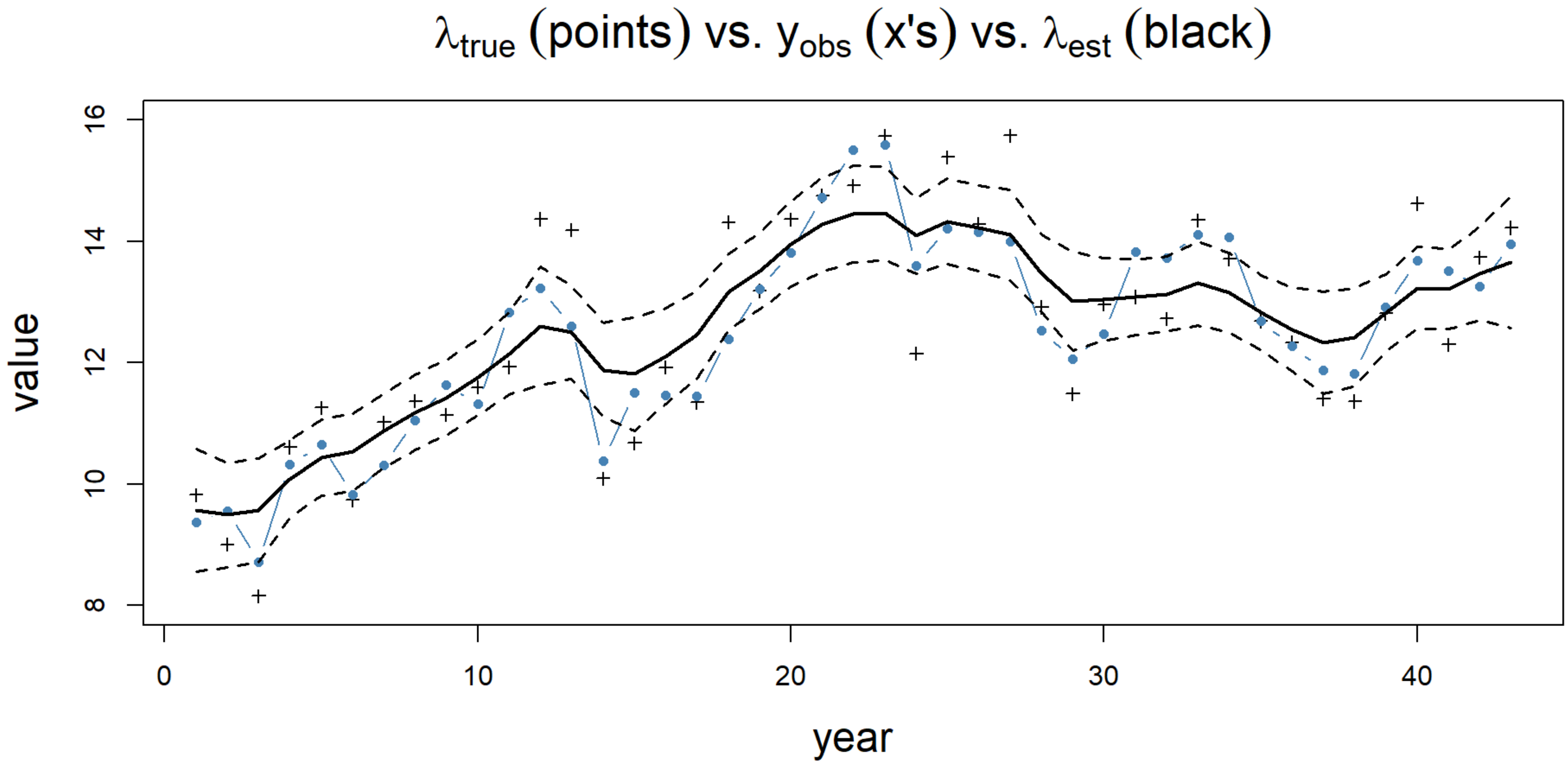
```
iter: 1  mgc: 8.21565e-15
outer mgc:  3.136849e-05
```

```
if (any(abs(final_gradient) > 0.001) || sdr$pdHess == FALSE) {
    message("Model did not converge: check results")
} else {
  message("Model diagnostics consistent with convergence")
}
```

```
[1] "Model fit consistent with convergence"
```

- Why are these reasonable things to check?

Fits vs. simulated data



How did we plot those results?

- First, look at the structure of the `sdreport()` output

```
str(sdr)
```

```
List of 10
 $ value          : num(0) 
 $ sd             : num(0) 
 $ cov            : logi[0 , 0 ] 
 $ par.fixed      : Named num [1:3] -0.5022 0.0758 9.5702
  ..- attr(*, "names")= chr [1:3] "ln_sd_rw" "ln_sd_obs" "lam0"
 $ cov.fixed      : num [1:3, 1:3] 0.1325 -0.0336 -0.0494 -0.0336 0.0286 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "ln_sd_rw" "ln_sd_obs" "lam0"
  .. ..$ : chr [1:3] "ln_sd_rw" "ln_sd_obs" "lam0"
 $ pdHess         : logi TRUE
 $ gradient.fixed : num [1:3] -7.01e-07 -3.14e-05 -4.76e-06
 $ par.random     : Named num [1:43] 9.57 9.49 9.57 10.09 10.44 ...
  ..- attr(*, "names")= chr [1:43] "lams" "lams" "lams" "lams" ...
 $ diag.cov.random: num [1:43] 0.515 0.442 0.435 0.323 0.319 ...
```

How did we plot those results?

- How do we get 95% confidence intervals?

```
mle = sdr$par.random
upper_ci = mle + 1.96 * sdr$diag.cov.random
lower_ci = mle - 1.96 * sdr$diag.cov.random
```

Break

Part II: dealing with missing data

- Two minor changes to our `rw.cpp` file allows us to deal with missing data

Dealing with missing data

- First, we write a custom function to detect NA values and place it at the top of the `rw.cpp`:

```
#include <TMB.hpp>
template<class Type>
// write a custom function to deal with NAs:
bool is_NA(Type x){
  return R_IsNA(asDouble(x));
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(y_obs);                          // observed data
  PARAMETER(ln_sd_rw);                         // log(sd) process
  PARAMETER(ln_sd_obs);                        // log(sd) observation
  PARAMETER(lam0);                             // initial state
...
```

Dealing with missing data

- Next, we correct the likelihood by adding an if-statement:

```
1  ...
2   // random effects:
3    jnll -= dnorm(lams(0),lam0,sd_rw,true);           // pr(inital state)
4    for(int i=1; i<n_year; i++){
5      jnll -= dnorm(lams(i),lams(i-1),sd_rw,true);    // pr(subsequent states)
6    }
7
8    // likelihood:
9    for(int i=0; i<n_year; i++){
10     if(!is_NA(y_obs(i))){
11       jnll -= dnorm(y_obs(i),lams(i),sd_obs,true); // pr(observations)
12     }
13   }
14 ...
```

- Note these aren't the true line numbers in `rw.cpp`

Remove some data and try it out

```
compile("rw.cpp")
```

```
Warning: 4 external pointers will be removed
Note: Library 'rw.dll' was unloaded.
```

```
[1] 0
```

```
# create dynamically linked library:
dyn.load(dynlib("rw"))

# make some funky data:
y_obs2 = y_obs
y_obs2[31:35] = NA

# create a tagged data list:
data <- list(y_obs = y_obs2)

# create a tagged parameter list w/ start values:
parameters <- list(ln_sd_rw = 0,
                   ln_sd_obs = 0,
                   lam0 = 0,
                   lams = rep(0, length(data$y_obs))
)

```

Remove some data and try it out

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

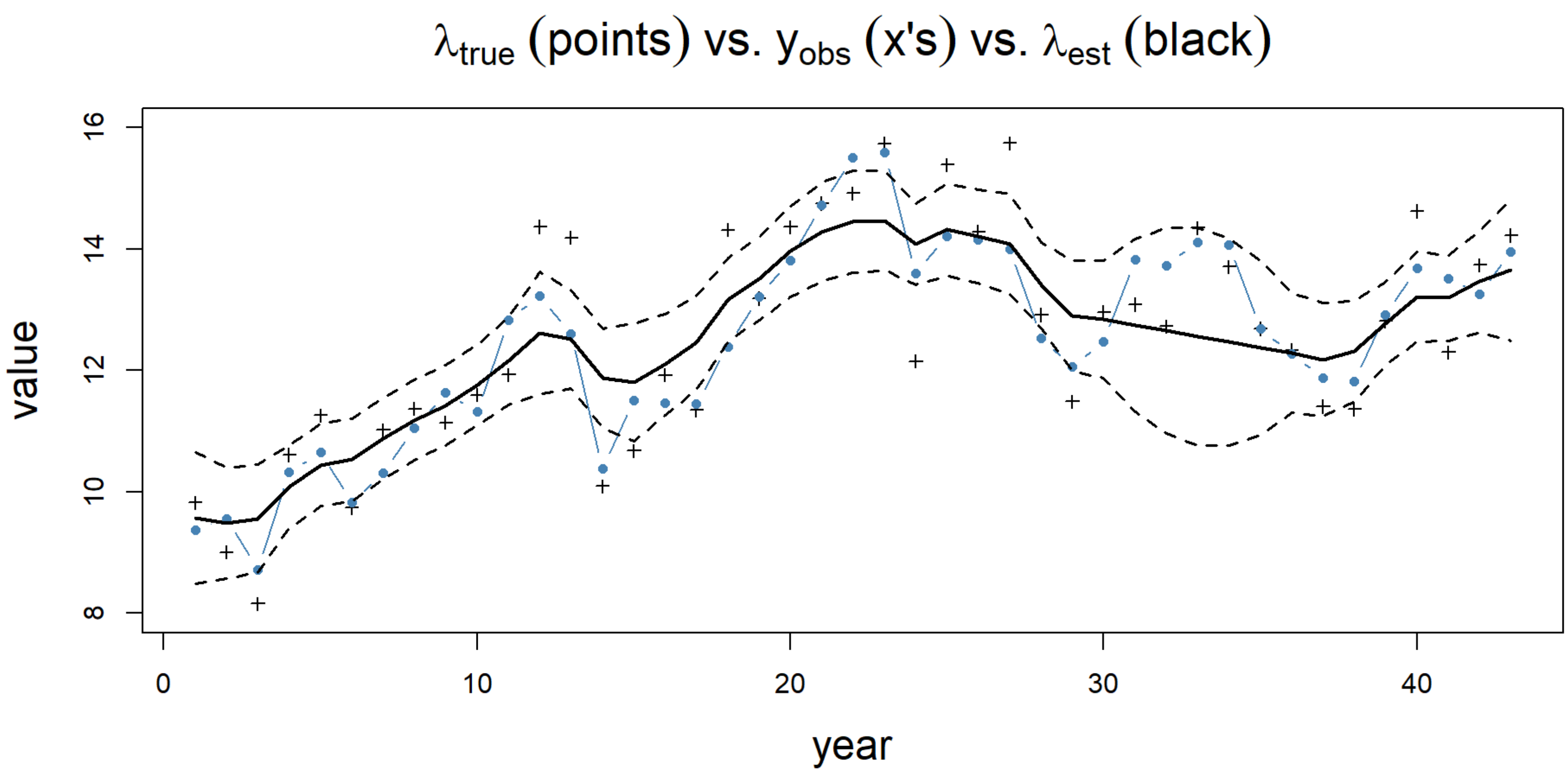
```
Optimizing tape... Done
iter: 1  value: 118.6808 mgc: 15.7428 ustep: 1
iter: 2  mgc: 1.065814e-14
iter: 1  mgc: 1.065814e-14
Matching hessian patterns... Done
outer mgc:  35.42793
iter: 1  value: 138.4644 mgc: 1.200569 ustep: 1
iter: 2  mgc: 8.187895e-16
iter: 1  mgc: 8.187895e-16
outer mgc:  10.50914
iter: 1  value: 118.3067 mgc: 2.883293 ustep: 1
iter: 2  mgc: 1.720846e-15
iter: 1  mgc: 1.720846e-15
outer mgc:  14.04101
iter: 1  value: 103.1652 mgc: 1.775114 ustep: 1
```

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

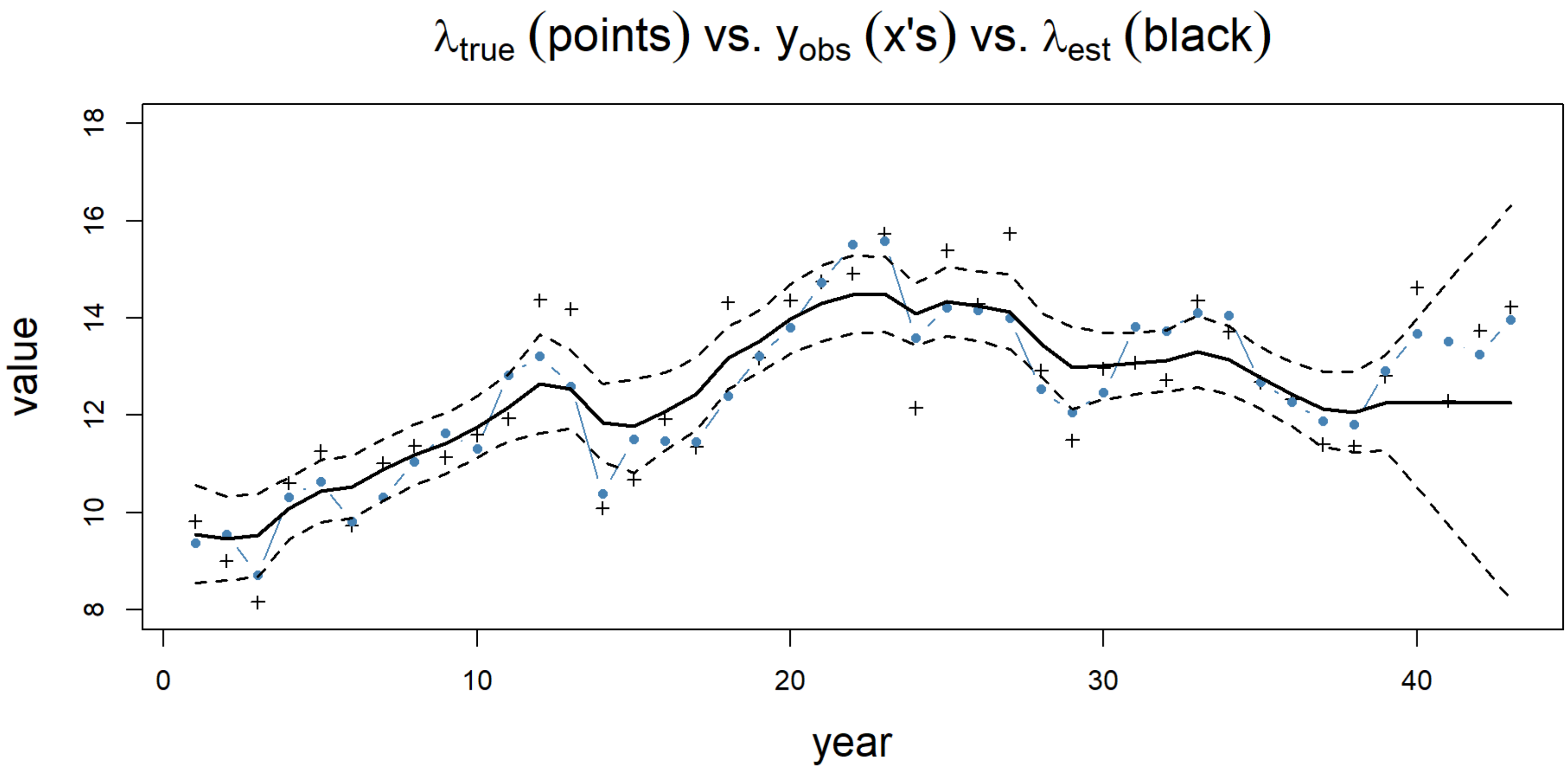
```
iter: 1  mgc: 7.674417e-15
outer mgc:  6.21457e-06
iter: 1  value: 77.98846 mgc: 0.003107419 ustep: 1
iter: 2  mgc: 6.550316e-15
outer mgc:  0.01087376
```

```
iter: 1  value: 77.92398 mgc: 0.00311364 ustep: 1
iter: 2  mgc: 7.327472e-15
outer mgc:  0.01086775
iter: 1  value: 77.967 mgc: 0.003107419 ustep: 1
iter: 2  mgc: 5.606626e-15
outer mgc:  0.04381543
iter: 1  value: 77.94548 mgc: 0.00311364 ustep: 1
iter: 2  mgc: 6.661338e-15
outer mgc:  0.04383615
iter: 1  value: 77.95622 mgc: 0.002488365 ustep: 1
```

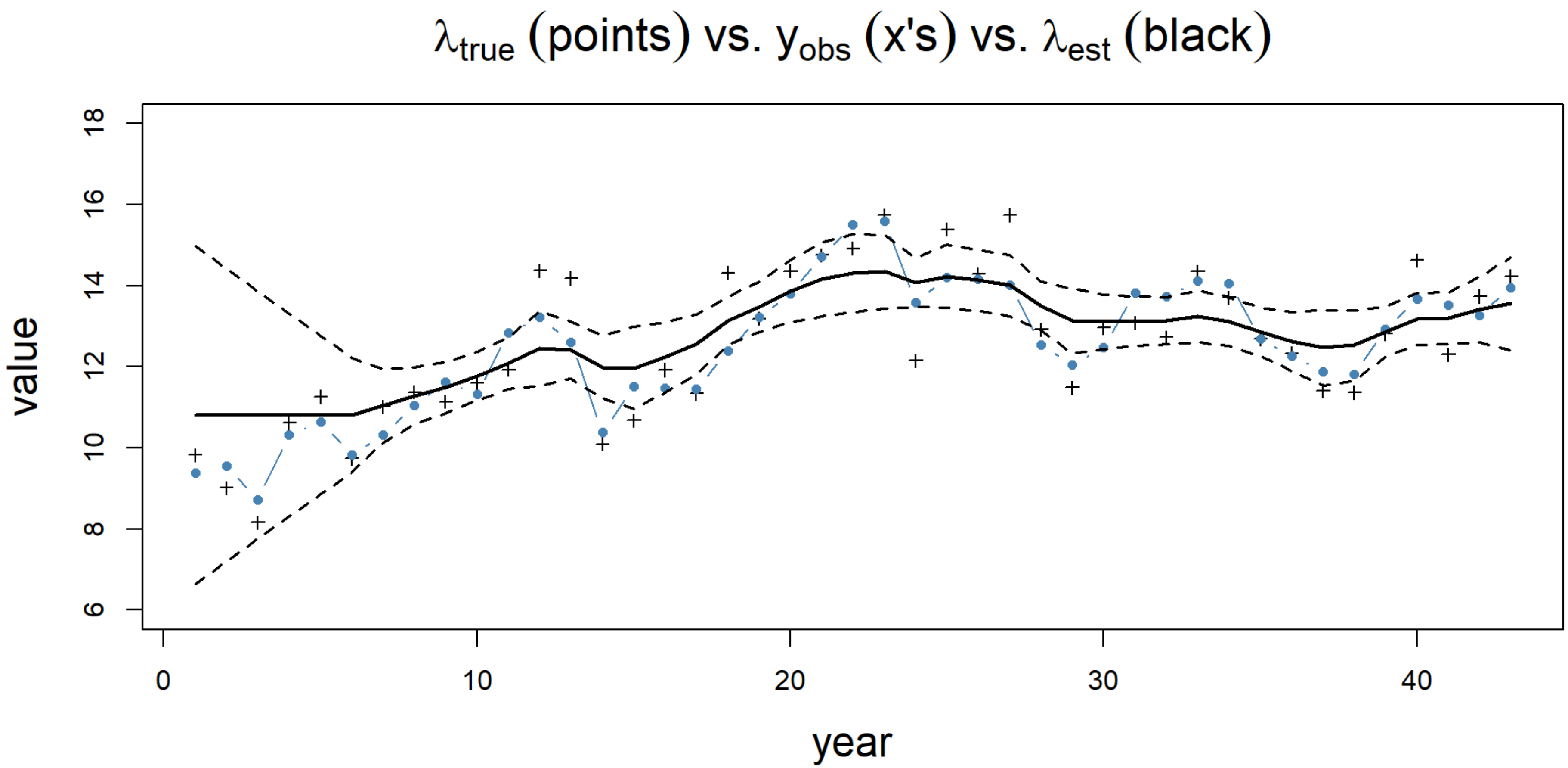
Visualize the hierarchical model fit - remove data years 31-35



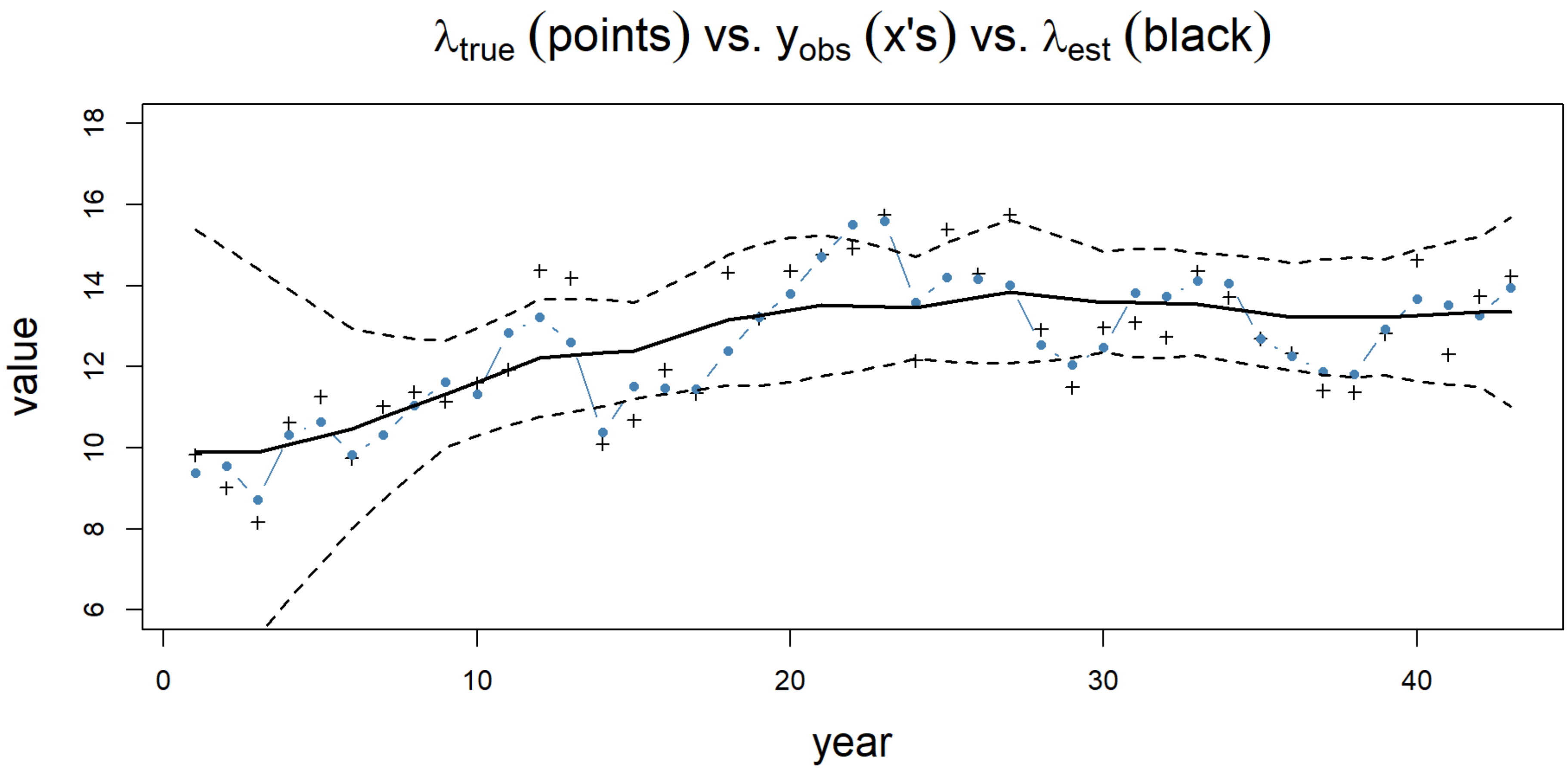
Visualize the hierarchical model fit - remove last 3 years



Visualize the hierarchical model fit - remove first 5 years



Visualize the hierarchical model fit - data every third year



Concluding remarks

- We have skipped over a lot of important theory and math.
- If n was large and we repeated this simulation experiment many times what do we expect?

Concluding remarks

- We have skipped over a lot of important theory and math.
- If *n* was large and we repeated this simulation experiment many times what do we expect to happen?
- The maximum likelihood estimator is *consistent*.
- $\hat{\theta}_{\mathrm{mle}} \xrightarrow{\mathrm{p}} \theta_0$.
- Most of what we have done here can be done entirely within TMB using a special `SIMULATE{}` block.
- There are other ways to do this.

Useful references

- Royle and Dorazio. 2008. Hierarchical modeling and inference in Ecology.
- Kéry and Schaub. 2012. Bayesian population analysis: A hierarchical perspective
- Holmes, Scheuerell, and Ward. 2021. Applied Time Series Analysis for Fisheries and Environmental Sciences. https://atsa-es.github.io/atsa-labs/index.html#authors