Autoregressive models and simulation

Ben Bolker

16:11 18 September 2015

Topics

- semi-simple autoregressive models
- tips for statistical computing
- parametric bootstrapping
- the geometry of optimization
- the geometry of statistical inference

Semi-simple AR models

Brian Dennis and Taper (1994; B. Dennis et al. 2006)

```
library("ggplot2")
theme_set(theme_bw())
library("reshape2")
```

The basic Ricker model of population dynamics:

$$\begin{aligned} N_{t+1} &= N_t \exp(a + bN_t + \sigma Z_t) \\ Z_t &\sim N(0,1) \end{aligned}$$

Makes ecological sense (usually expressed in ecology as $N_{t+1} = rN_t \exp(-cN_t)$. Z_t incorporates *process error*. Log-transforming makes it into a *linear autoregressive* model (linearity in the *parameters* a, b).

$$X_{t+1} = X_t + a + be^{X_t} + \sigma Z_t$$

$a = 0, b = 0$	zero-drift Brownian motion
$a \neq 0, b = 0$	Brownian motion with drift (exp growth/decay)
$a \neq 0, b \neq 0$	density-dependent growth
$a \neq 0, b < 0$	regulation

Data from Dennis and Taper (Yellowstone grizzlies):

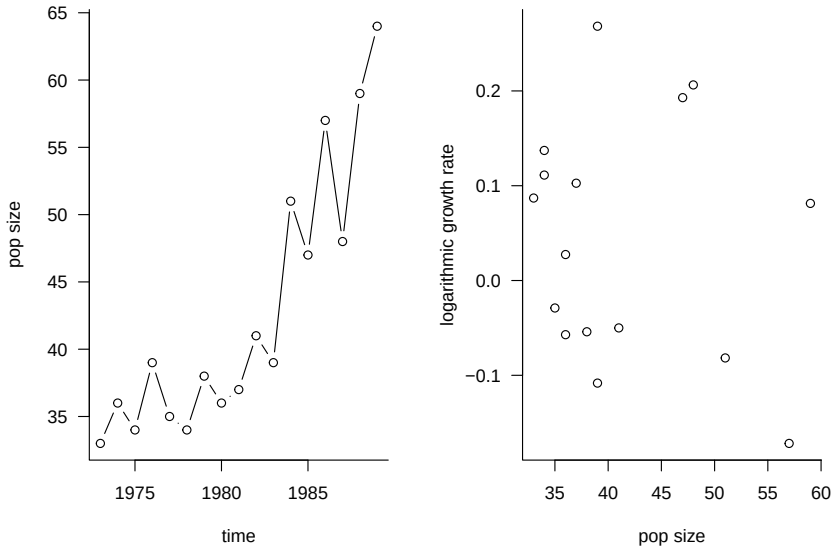
```
grizzly <- data.frame(n = c(33, 36, 34, 39, 35,
    34, 38, 36, 37, 41, 39, 51, 47, 57, 48, 59,
    64), t = 1973:1989)
```

Lagging variables in R is a little bit clunky; we have to drop the first and last elements respectively. `head(x,-1)` is an alternative to `x[-n]`. `lag()` doesn't work the way you think it does (!)

```
r <- log(grizzly$n[-1]/grizzly$n[-length(grizzly$n)])
```

Look at the data:

```
par(las = 1, bty = "l", mfrow = c(1, 2))
plot(n ~ t, data = grizzly, type = "b", xlab = "time",
    ylab = "pop size")
plot(grizzly$n[-length(grizzly$n)], r, xlab = "pop size",
    ylab = "logarithmic growth rate")
```



Transform data and fit:

```
## R tip: use with() to simplify extraction
## from data frames -- **never** attach()
dd <- with(grizzly, data.frame(xlag1 = log(n[-length(n)]),
    x = log(n[-1])))
m2 <- lm(x ~ offset(xlag1) + exp(xlag1), data = dd)
## equivalently: lm(x-xlag1~exp(xlag1),data=dd)
## (offset may be easier to read)
s2 <- sigma(m2)^2
## D&T use MLE rather than unbiased estimate of
## variance parameters:
var.unbiased <- function(m) sigma(m)^2 * df.residual(m)/nobs(m)
## parameters plus variance:
c(coef(m2), var = var.unbiased(m2))
```

```
##  (Intercept)   exp(xlag1)          var
##  0.141462270 -0.002411176  0.014524403
```

```
## test statistic
coef(summary(m2))[2, "t value"]
```

```
## [1] -0.6041122
```

```
## R tip: use update() when possible to
## simplify code
m1 <- update(m2, . ~ offset(xlag1))
c(coef(m1), var.unbiased(m1))
```

```
## (Intercept)
##  0.04139847  0.01490303
```

Parametric bootstrapping:

- for *confidence intervals*: simulate from the full model and record values. Compute quantiles.
- for *hypothesis tests*: simulate from the null model; fit the null model and the full model

Simulation

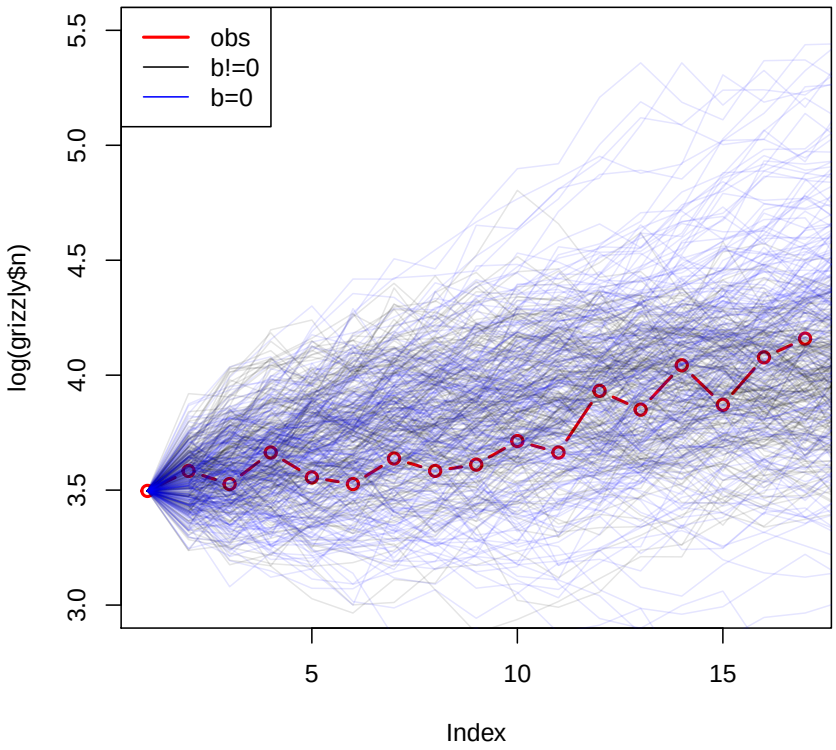
I thought I could use the `simulate()` method in R, but as it turns out I can't, so I have to define my own:

```
## simulate new data, add starting value
simfun <- function(fit, x0 = log(grizzly$n[1])) {
    cc <- coef(fit)
    a <- cc[1]
    ## make the function work for either constant
    ## or regression model
    b <- if (length(cc) == 2)
        coef(fit)[2] else 0
    sigma <- sqrt(var.unbiased(fit))
    ## as usual, half the code is setup: the next
    ## three lines are the actual meat
    res <- c(x0, numeric(nobs(fit) + 1))
    for (i in 2:length(res)) {
        res[i] <- res[i - 1] + a + b * exp(res[i -
            1]) + rnorm(1, sd = sigma)
    }
    res
}
```

Test simulation function:

```
sim2 <- replicate(200, simfun(m2))
sim1 <- replicate(200, simfun(m1))
```

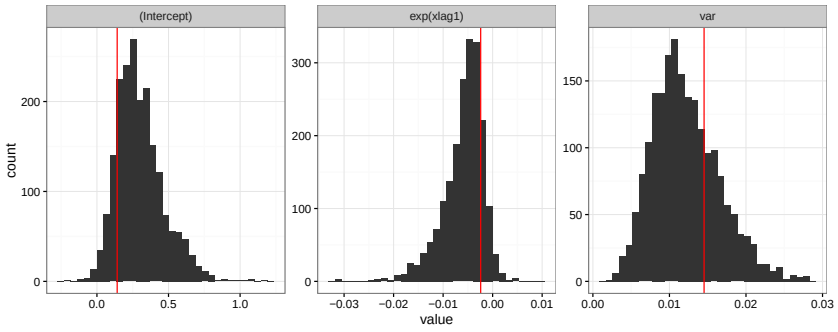
```
plot(log(grizzly$n), type = "b", ylim = c(3, 5.5),
    lwd = 2, col = 2)
## convenient to plot lots of lines with
## semi-transparent (adjustcolor(...,'alpha'))
## colour
matlines(sim2, col = adjustcolor("black", alpha = 0.1),
    lty = 1)
matlines(sim1, col = adjustcolor("blue", alpha = 0.1),
    lty = 1)
legend("topleft", lty = 1, col = c("red", "black",
    "blue"), lwd = c(2, 1, 1), c("obs", "b!=0",
    "b=0"))
```




```
## re-fit model with new data
fitfun <- function(x, orig_fit) {
    dd <- data.frame(xlag1 = x[-length(x)], x = x[-1])
    update(orig_fit, data = dd)
}
## test:
all.equal(fitfun(log(grizzly$n), m2), m2)
```

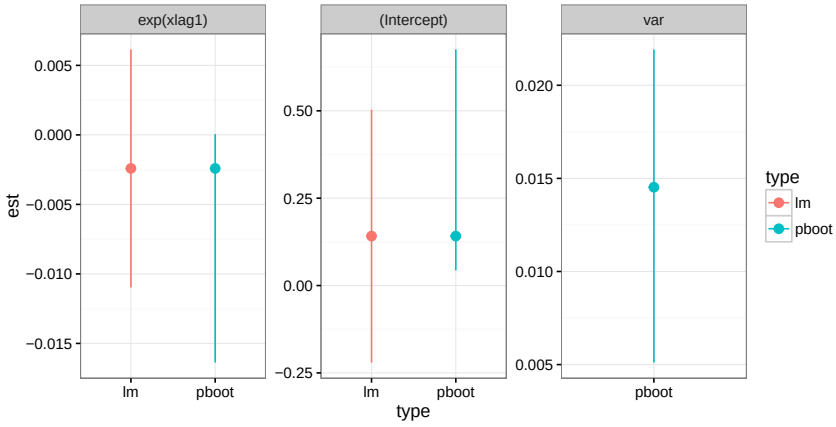
```
## [1] TRUE
```

```
## summarize output
sumfun <- function(fit) {
    c(coef(fit), var = var.unbiased(fit))
}
## combine all of the above
bootfun <- function(fit) {
    sumfun(fitfun(simfun(fit), fit))
}

set.seed(101)  ## for reproducibility
## t() transposes results
v <- t(replicate(2000, bootfun(m2)))

## using ggplot, a fancy but powerful plotting
## tool:
mv <- melt(as.data.frame(v))  ## restructure data in 'long' format
obsvals <- data.frame(variable = c("(Intercept)",
    "exp(xlag1)", "var"), value = sumfun(m2))
ggplot(mv, aes(value)) + facet_wrap(~variable,
    scale = "free") + geom_histogram(bins = 30) +
    geom_vline(data = obsvals, colour = "red",
        aes(xintercept = value))
```




```
## compare estimates
tfun <- function(x, type, est) {
    colnames(x) <- c("lwr", "upr")
    x <- data.frame(type, variable = rownames(x),
        est, x)
    rownames(x) <- NULL
    return(x)
}
bootconf <- t(apply(v, 2, quantile, c(0.025, 0.975)))
allCI <- rbind(tfun(confint(m2), "lm", coef(m2)),
    tfun(bootconf, "pboot", sumfun(m2)))
```

```
ggplot(allCI, aes(type, est, ymin = lwr, ymax = upr,
    colour = type)) + geom_pointrange() + facet_wrap(~variable,
    scale = "free")
```



Geometry of optimization (Press et al. 1994)

- need to find point estimate (best parameters), *or* get close
- frequentist MLE, Bayesian maximum *a posteriori* (mode of posterior distribution)
- geometric analogies; hill-climbing
- simplest cases: use derivatives
  - Newton-Raphson (good in theory, but fragile)
  - conjugate gradients (`method="CG"` in R `optim`)
  - quasi-Newton (`method="BFGS"` in R `optim`)
- derivatives can be expensive, fragile
  - Nelder-Mead (`method="Nelder-Mead"`)
  - BOBYQA (`nloptr` package)
- *complications*: noisy surfaces
  - stochastic *global* optimizers: simulated annealing, genetic algorithms, differential evolution
- *complications*: multiple peaks
  - repeated starts
  - stochastic global optimization
- similar geometric issues apply to MCMC

Geometry of inference (Bolker 2008, ch. 6-7)

- Wald intervals: quadratic
- profile confidence intervals
- marginal intervals

References

Bolker, Benjamin M. 2008. *Ecological Models and Data in R*. Princeton, NJ: Princeton University Press.

Dennis, B., J. M Ponciano, S. R Lele, M. L Taper, and D. F Staples. 2006. "Estimating Density Dependence, Process Noise, and Observation Error." *Ecological Monographs* 76 (3): 323–41.

Dennis, Brian, and Mark L. Taper. 1994. "Density Dependence in Time Series Observations of Natural Populations: Estimation and Testing." *Ecological Monographs* 64 (2): 205–24. doi:10.2307/2937041.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1994. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press. `http://www.nr.com/oldverswitcher.html`.