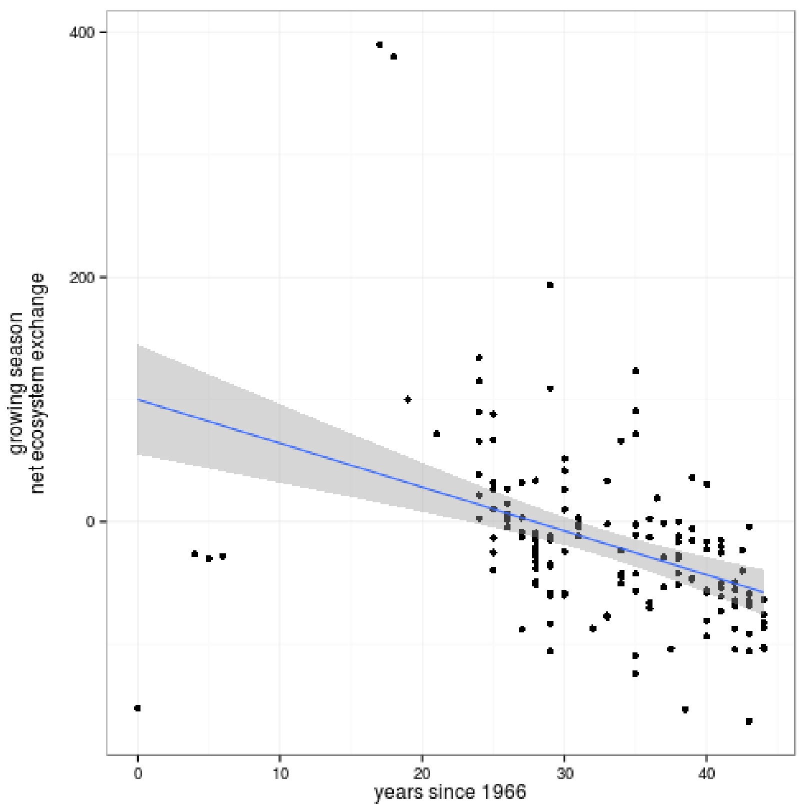
Statistical analysis 1: linear models, wheel-building and randomization

2014-08-04 01:39:44

Linear models

```
tdat <- read.csv("tundra.csv",na.strings=c("-","NA"))
tdat <- transform(tdat,cYear=Year-min(Year))
ggplot(tdat,aes(cYear,GS.NEE))+geom_point()+geom_smooth(method="lm")+
    labs(x="years since 1966",y="growing season\nnet ecosystem exchange\n")
```




```
lm0 <- lm(GS.NEE~cYear,data=tdat,na=na.exclude)
lm1 <- lm(GS.GPP~cYear,data=tdat,na=na.exclude)
```

Wheel-building

- Almost all statistical methods involve *optimizing* (min/max) a *loss function* (e.g. sum of squares, log-likelihood)

- When should you make up your own statistical methods *vs.* taking something off the shelf?
- **advantages**: better understanding, customization, flexibility, fame & glory
- **disadvantages**: inefficiency (statistical & computational & programming); having to explain yourself
- Basic tools (big overlap with Bolker (2008) chap. 5)
  - writing functions in R
  - optimization in R
  - replicate and factorial simulation runs
  - stochastic simulation

Randomization

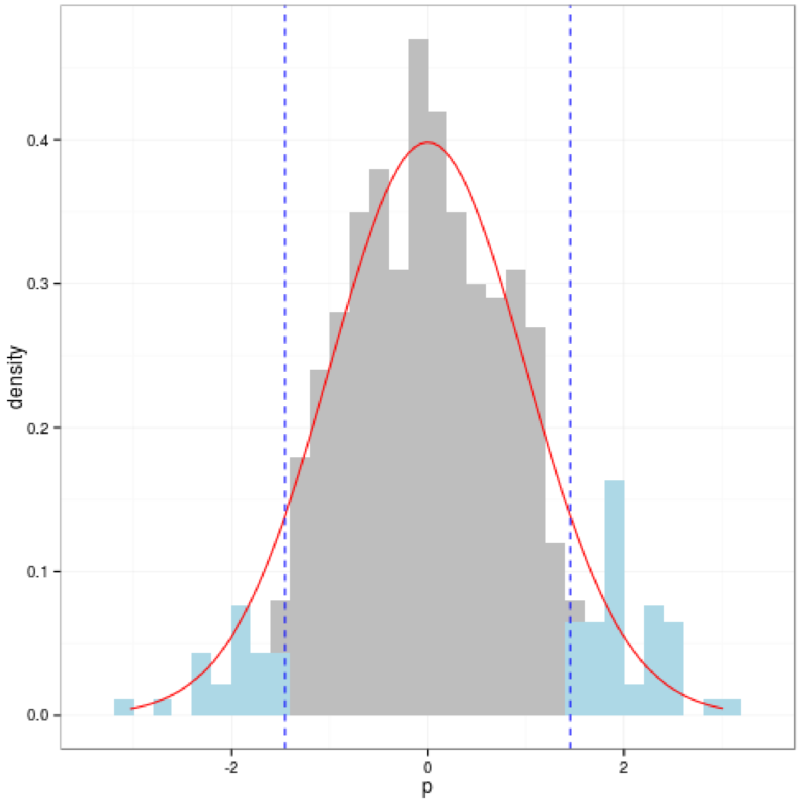
Permutation tests

- basic idea: simulate the null hypothesis
- *if* data are independent samples, can just scramble the order of the response variable
- `sample()` does this:

```
sim.data <- transform(orig.data,response=sample(response))
```

- need to replicate and store values of the test statistic
- compute fraction of the values >= than the observed statistic:

```
mean(obs.stat >= perm.stat)
```



- in principle any test statistic will do, may be easiest to use the equivalent of the classical test
- `str()`, `summary()` are helpful for digging out test statistics
- functions for (1) resampling data, (2) fitting model, (3) extracting summary statistics (some might be combined)
- only gives p values – not confidence intervals
- tricky if data aren't independent – e.g. may need to permute blocks instead
- the `lmPerm` package implements permutation tests for linear models (`lmp` function)
- classical tests are often *extremely* good (i.e. permutation tests weren't that needed after all)

```
m1 <- lm(GS.GPP~cYear,data=tdat)
m1P <- lmp(GS.GPP~cYear,data=tdat,Ca=0.001,maxIter=1e6)
```

```
##     Estimate Pr(>|t|)
## lm    -3.016   0.1508
## lmp   -3.016   0.1483
```

- difficulties with extremely small p values (e.g. bioinformatics)

Bootstrapping

- basic idea: resample existing data *with replacement*
- *if* data are independent samples, sampling with replacement is like doing a nonparametric "simulation"
- `sample()` can sample with replacement too, but need to sample *entire* data set (maintain relationship between predictor & response variables):
- need to replicate and store values of the estimates (usually `coef()`)
- functions for (1) resampling data, (2) fitting model, (3) extracting coefficients
- tricky if data aren't independent – *block bootstrapping*
- bootstrap confidence intervals: simplest possibilities are (1) Normal approximation (2) percentile/quantile
- the `boot` package implements many possibilities, but I find it clunky
- as with permutation, bootstrap CI are often similar to classical CIs

Parametric bootstrapping/posterior predictive simulation

- for goodness-of-fit testing and computing prediction intervals
- like permutation tests, but (1) simulate under the fitted model, not the alternative; (2) more interesting summary statistics
- for simple (linear or weakly nonlinear) predictions, can calculate mean and variance directly or approximate via delta method (Bolker 2008; Hilborn and Mangel 1997)
- for most R models can resample parameters via

```
MASS::mvrnorm(n.sim,
              mu=coef(fitted.model),
              Sigma=vcov(fitted.model))
```

  or via `arm::sim()`.

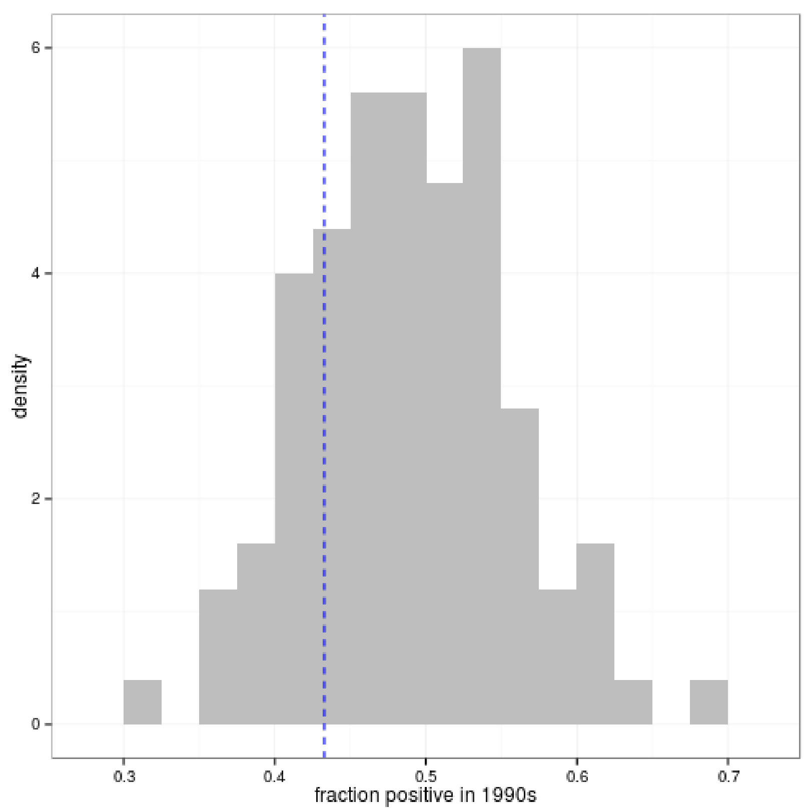
- *Example*: what fraction of the values were positive between 1990 and 2000?

```
nineties <- subset(tdat,Year>=1990 & Year<2000)
nval <- nrow(nineties)
rfun <- function() {
    ss <- arm::sim(lm0,1)  ## get one random value
    vals <- rnorm(nval,mean=ss@coef[1]+ss@coef[2]*nineties$cYear,
                  sd=ss@sigma)
    mean(vals>0)
}
mvals <- rdply(100,rfun())
summary(mvals$V1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.316   0.442   0.484   0.489   0.539   0.695
```

```
(obs.val <- mean(nineties$GS.NEE>0,na.rm=TRUE))
```

```
## [1] 0.4328
```

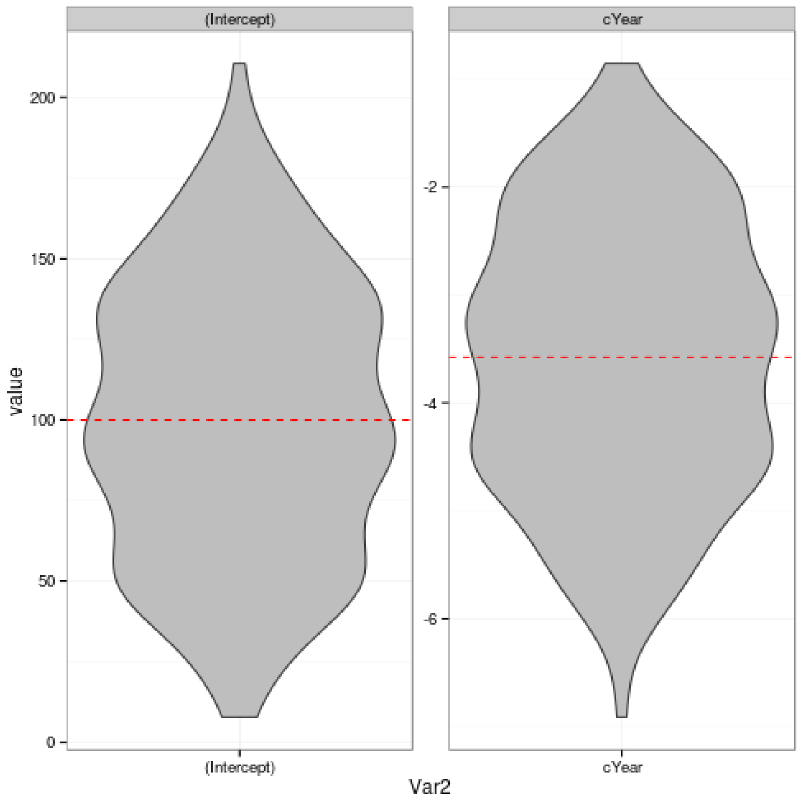


- easy with MCMC approaches
- unfortunately the `simulate()` method only incorporates process/measurement error, while `sim()` only incorporates estimation error

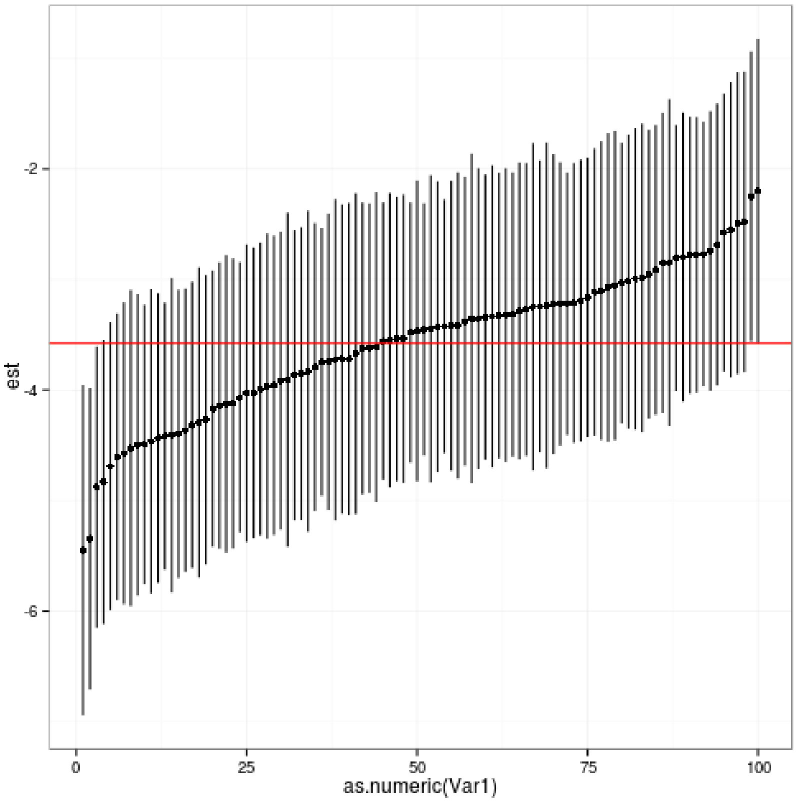
Method assessment and testing

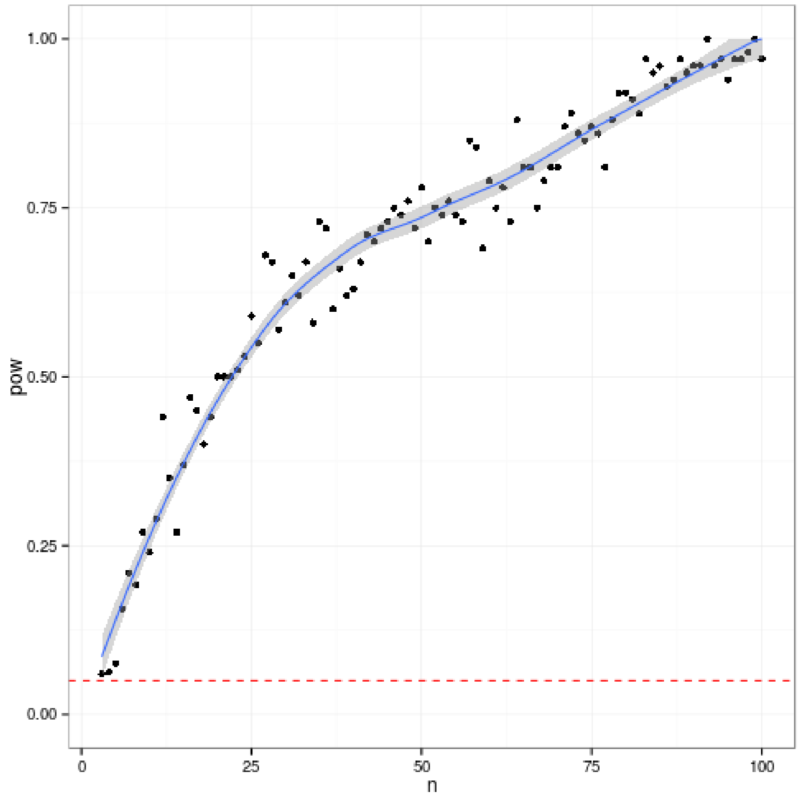
precision	**accuracy**
variance	bias
coefficient of variation	type I error rate
CI width	type I error rate
power	*coverage*
mean squared error	mean squared error

- power/performance analysis
  - loop over variable(s) of interest (sample size, effect size, etc.)
    - loop over replicates
      - simulate data
      - analyze data (= fit model)
      - store results (parameter estimates, confidence intervals, p value ...)
  - analyze results
- *violin plot* (check for bias):



- *caterpillar plot* (check for coverage):


est
-2
-4
-6
0
25
50
75
100
as.numeric(Var1)



Tips for simulation studies

- use `try()` in case things break; e.g. `try(x,silent=TRUE); if (inherits(x,"try-error")) return(NA)`
- set random number seeds! possibly sequentially
- organize factorial experiment results as multi-dimensional arrays (with well-named dimensions)
- use `apply()` to collapse across dimensions (e.g. to get power, coverage, mean bias, MSE)
- use `reshape2::melt` to get from array to long form data

References

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

Hilborn, R., and M. Mangel. 1997. *The Ecological Detective : Confronting Models with Data*. Princeton, New Jersey, USA: Princeton University Press.