



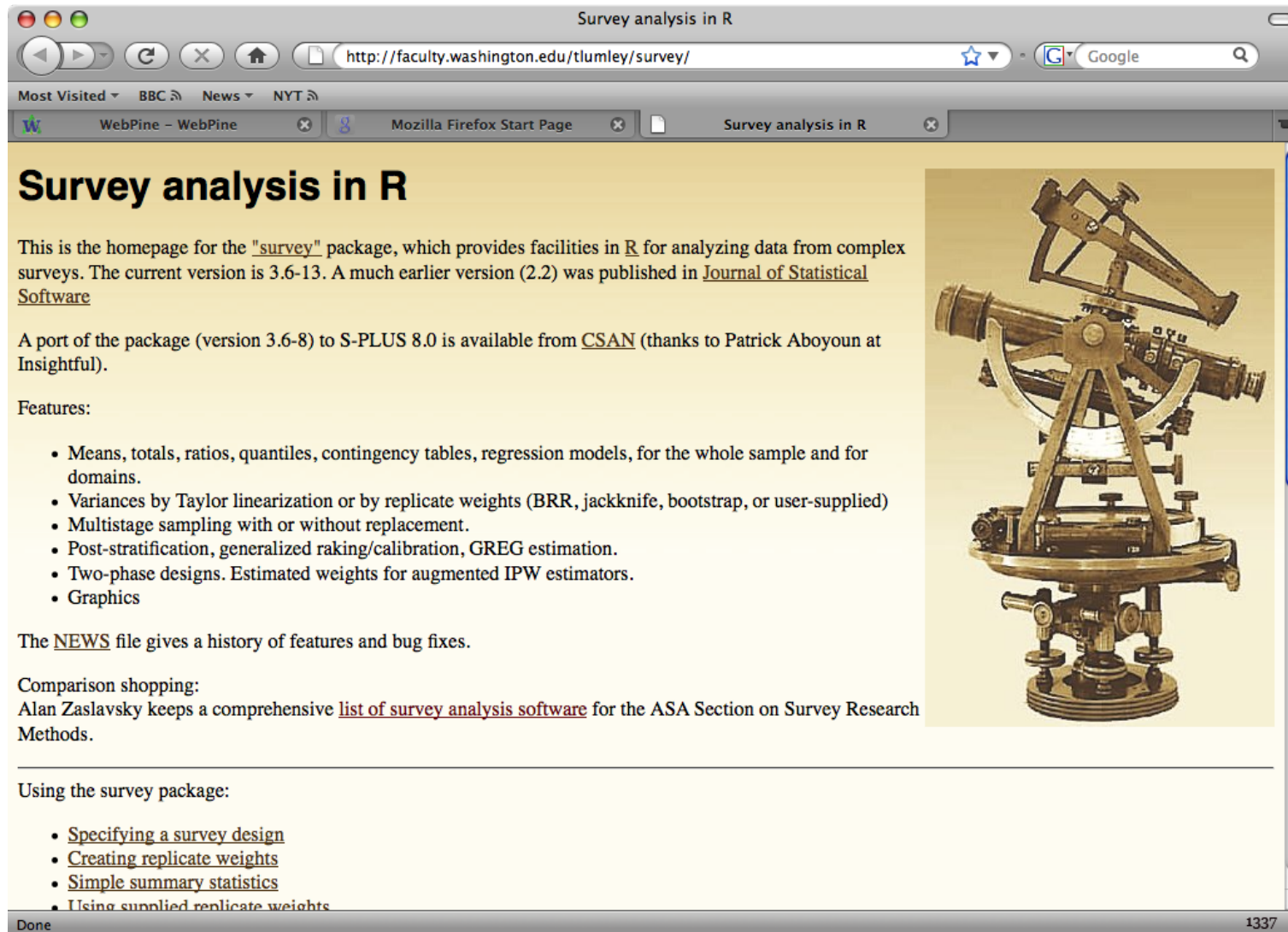
The survey package in R

Thomas Lumley

Biostatistics
University of Washington

WSS short course — 2009-3-24

R Survey package



Survey analysis in R

<http://faculty.washington.edu/tlumley/survey/>

Survey analysis in R

This is the homepage for the "[survey](#)" package, which provides facilities in R for analyzing data from complex surveys. The current version is 3.6-13. A much earlier version (2.2) was published in [Journal of Statistical Software](#)

A port of the package (version 3.6-8) to S-PLUS 8.0 is available from [CSAN](#) (thanks to Patrick Aboyoun at Insightful).

Features:

- Means, totals, ratios, quantiles, contingency tables, regression models, for the whole sample and for domains.
- Variances by Taylor linearization or by replicate weights (BRR, jackknife, bootstrap, or user-supplied)
- Multistage sampling with or without replacement.
- Post-stratification, generalized raking/calibration, GREG estimation.
- Two-phase designs. Estimated weights for augmented IPW estimators.
- Graphics

The [NEWS](#) file gives a history of features and bug fixes.

Comparison shopping:
Alan Zaslavsky keeps a comprehensive [list of survey analysis software](#) for the ASA Section on Survey Research Methods.

Using the survey package:

- [Specifying a survey design](#)
- [Creating replicate weights](#)
- [Simple summary statistics](#)
- [Using supplied replicate weights](#)

Done 1337

<http://faculty.washington.edu/tlumley/survey/>

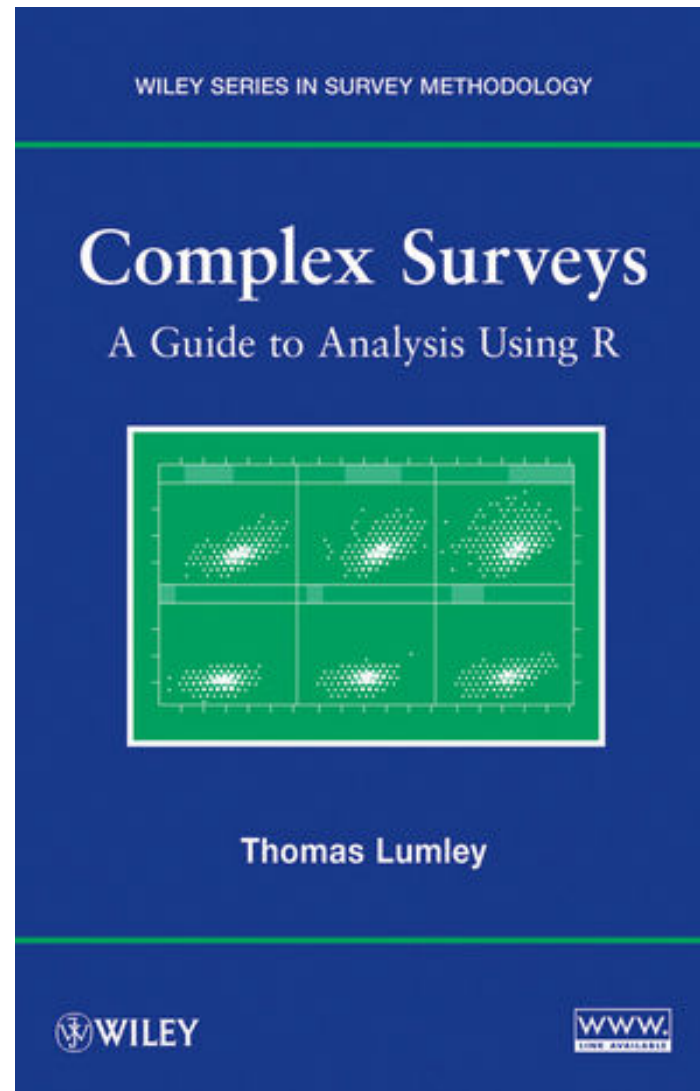
R Survey package

Version 3.21-1 is current, containing approximately 11000 lines of interpreted R code.

Version 2.3 was published in Journal of Statistical Software.

Major changes since then: finite population corrections for multistage sampling and PPS sampling, calibration and generalized raking, tests of independence in contingency tables, better tables of results, simple two-phase designs, loglinear models, ordinal regression models, simple multivariate analyses, survival curves

The book of the package



<http://faculty.washington.edu/tlumley/svybook/>

Design principles

- Ease of maintenance and debugging by code reuse
- Speed and memory use not initially a priority: don't optimize until there are real use-cases to profile.
- Rapid release, so that bugs and other infelicities can be found and fixed.
- Emphasize features that look like biostatistics (regression, calibration, survival analysis, exploratory graphics, two-phase sampling)

Intended market

- Methods research (because of programming features of R)
- Teaching (because of cost, use of R in other courses)
- Secondary analysis of national surveys (regression features, R is familiar to non-survey statisticians)
- Two-phase designs in epidemiology

Outline

- Describing survey designs: `svydesign()`
- Database-backed designs
- Summary statistics: mean, total, quantiles, design effect
- Tables of summary statistics, domain estimation.
- Contingency tables: `svychisq()`, `svyloglin()`
- Graphics: histograms, hexbin scatterplots, smoothers.
- Regression modelling: `svyglm()`, `svyolr()`,
- Calibration
- Multiply-imputed data
- Two-phase designs, PPS sampling.

Objects and Formulas

Collections of related information should be kept together in an object. For surveys this means the data and the survey meta-data.

The way to specify variables from a data frame or object in R is a formula

```
~a + b + I(c < 5*d)
```

The survey package **always** uses formulas to specify variables in a survey data set.

Basic estimation ideas

Individuals are sampled with known probabilities π_i from a population of size N to end up with a sample of size n . The population can be a full population or an existing sample such as a cohort.

We write $R_i = 1$ if individual i is sampled, $R_i = 0$ otherwise

The design-based inference problem is to estimate what any statistic of interest would be if data from the whole population were available.

Basic estimation ideas

For a population total this is easy: an unbiased estimator of

$$T_X = \sum_{i=1}^N x_i$$

is

$$\hat{T}_X = \sum_{i:R_i=1} \frac{1}{\pi_i} X_i$$

Standard errors follow from formulas for the variance of a sum: main complication is that we do need to know $\text{cov}[R_i, R_j]$.

Basic estimation ideas

The general Horvitz–Thompson formula is

$$\widehat{\text{var}} = \sum_{i,j=1}^n \check{\Delta}_{ij} \check{x}_i \check{x}_j$$

where Δ is the covariance of sampling indicators and the check indicates multiplication by the sampling weight (for x) or pairwise sampling weight (for Δ)

For multistage stratified/clustered designs this can be most easily computed by a recursive algorithm over the stages of sampling, in `svyrecvar()`.

Basic estimation ideas

Other statistics follow from sums: if the statistic on the whole population would solve the estimating equation

$$\sum_{i=1}^N U_i(\theta) = 0$$

then a design-based estimate will solve

$$\sum_{i:R_i=1} \frac{1}{\pi_i} U_i(\theta) = 0$$

Standard errors come from the delta method or from resampling (actually reweighting).

Theoretical details can become tricky, especially if $U_i(\cdot)$ is not a function of just one observation (eg Cox model)

Describing surveys to R

Stratified independent sample (without replacement) of California schools

```
data(api)
dstrat <- svydesign(id=~1, strata=~stype, weights=~pw,
  data=apistrat, fpc=~fpc)
```

- `stype` is a factor variable for elementary/middle/high school
- `fpc` is a numeric variable giving the number of schools in each stratum. If omitted we assume sampling with replacement
- `id=~1` specifies independent sampling.
- `apistrat` is the data frame with all the data.
- `pw` contains sampling weights ($1/\pi_i$). These could be omitted since they can be computed from the population size.

Note that all the variables are in `apistrat` and are specified as formulas.

Describing surveys to R

```
> dstrat
Stratified Independent Sampling design
svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
fpc = ~fpc)
> summary(dstrat)
Stratified Independent Sampling design
svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
fpc = ~fpc)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.02262 0.02262 0.03587 0.04014 0.05339 0.06623
Stratum Sizes:
E H M
obs 100 50 50
design.PSU 100 50 50
actual.PSU 100 50 50
Population stratum sizes (PSUs):
E M H
4421 1018 755
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```

Describing surveys to R

Cluster sample of school districts, using all schools within a district.

```
dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
```

- `dnum` is a (numeric) identifier for school district
- No stratification, so no `strata=` argument

```
> summary(dclus1)
1 - level Cluster Sampling design
With (15) clusters.
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.02954 0.02954 0.02954 0.02954 0.02954 0.02954
Population size (PSUs): 757
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```

Describing surveys to R

Two-stage sample: 40 school districts and up to 5 schools from each

```
dclus2 <- svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
```

- `dnum` identifies school district, `snum` identifies school
- `fpc1` is the number of school districts in population, `fpc2` is number of schools in the district.
- Weights are computed from `fpc1` and `fpc2`

```
> summary(dclus2)
2 - level Cluster Sampling design
With (40, 126) clusters.
svydesign(id = ~dnum + snum, fpc = ~fpc1 + fpc2, data = apiclus2)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.003669 0.037740 0.052840 0.042390 0.052840 0.052840
Population size (PSUs): 757
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```


Replicate weights

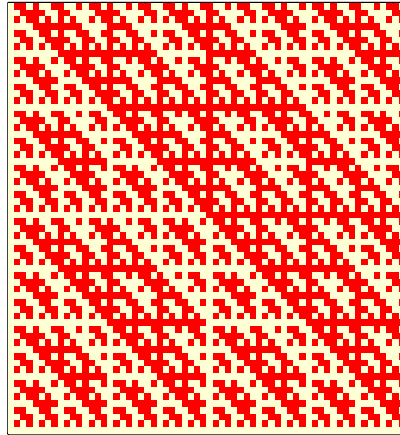
California Health Interview Survey has 50 sets of replicate weights instead of design information

```
chis_adult <- read.dta("adult.dta")
chis <- svrepdesign(variables=chis_adult[,1:418],
  repweights=chis_adult[,420:499],
  weights=chis_adult[,419], combined.weights=TRUE,
  type="other", scale=1, rscales=1)
```

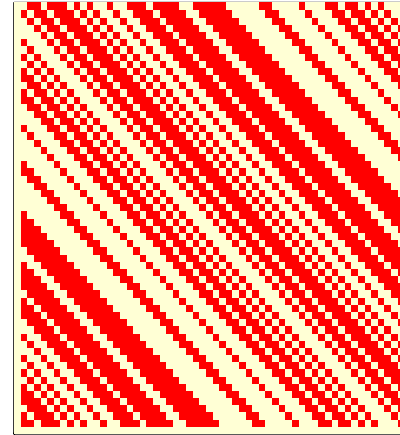
Can also create replicate weights with `as.svrepdesign()`, using jackknife, bootstrap, or BRR.

BRR: R knows Hadamard matrices

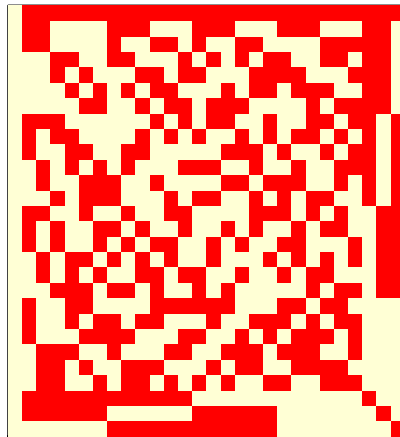
Sylvester: $64 = 2^6$



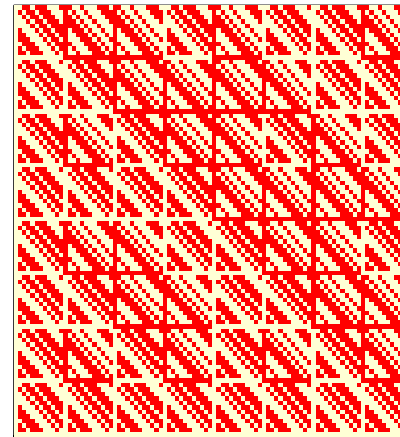
Paley: $60 = 59 + 1$



Stored: 28



Constructed: $96 = 2^3 \times (11 + 1)$



Describing surveys: database-based

```
library(RSQLite)
brfss <- svydesign(id=~X_PSU, strata=~X_STATE, weight=~X_FINALWT,
  data="brfss", dbtype="SQLite", dbname="brfss07.db",
  nest=TRUE)
```

The `data` argument is the name of a database table or view, `dbtype` and `dbname` specify the database.

Only the design meta-data are loaded into the design object. Other variables are temporarily loaded as needed when an analysis is run.

Can use any database with an ODBC, JDBC, or R-DBI interface: anything on Windows, SQLite, Postgres, Oracle.

BRFSS is about as large as a 1Gb laptop can handle with this approach: 430,000 records.

Summary statistics

`svymean`, `svytotal`, `svyratio`, `svyvar`, `svyquantile`

All take a formula and design object as arguments, return an object with `coef`, `vcov`, `SE`, `cv` methods.

Mean and total on factor variables give tables of cell means/totals.

Mean and total have `deff` argument for design effects and the returned object has a `deff` method.

Summary statistics

`confint()` works on most objects to give confidence intervals.

```
> svymean(~api00, dclus1, deff=TRUE)
```

```
      mean    SE    DEff
```

```
api00 644.169 23.542 9.3459
```

```
> svymean(~factor(stype), dclus1)
```

```
mean SE
```

```
factor(stype)E 0.786885 0.0463
```

```
factor(stype)H 0.076503 0.0268
```

```
factor(stype)M 0.136612 0.0296
```

Summary statistics

```
> svymean(~interaction(stype, comp.imp), dclus1)
mean SE
interaction(stype, comp.imp)E.No 0.174863 0.0260
interaction(stype, comp.imp)H.No 0.038251 0.0161
interaction(stype, comp.imp)M.No 0.060109 0.0246
interaction(stype, comp.imp)E.Yes 0.612022 0.0417
interaction(stype, comp.imp)H.Yes 0.038251 0.0161
interaction(stype, comp.imp)M.Yes 0.076503 0.0217
> svyvar(~api00, dclus1)
variance SE
api00 11183 1386.4
> svytotal(~enroll, dclus1, deff=TRUE)
total SE DEff
enroll 3404940 932235 31.311
```

Summary statistics

```
> mns <- svymean(~api00+api99,dclus1)
```

```
> mns
```

	mean	SE
api00	644.17	23.542
api99	606.98	24.225

```
> coef(mns)
```

api00	api99
644.1694	606.9781

```
> SE(mns)
```

api00	api99
23.54224	24.22504

Summary statistics

```
> vcov(mns)
```

```
          api00      api99
api00 554.2371 565.7856
api99 565.7856 586.8526
```

```
> cv(mns)
```

```
api00      api99
0.03654666 0.03991090
```

```
> confint(mns)
```

```
          2.5 %    97.5 %
api00 598.0275 690.3113
api99 559.4979 654.4583
```


Proportions

Proportions are just means, but there are better approaches to confidence intervals for proportions near 0 or 1.

`svyciprop()` also provides

- logistic regression interval
- ‘likelihood’ interval based on Rao–Scott test
- binomial ci based on effective sample size (Korn & Graubard 1998)
- arcsin–sqrt transformation.

Domain estimation

The correct standard error estimate for a subpopulation that isn't a stratum is not just obtained by pretending that the subpopulation was a designed survey of its own.

However, the `subset` function and `"["` method for survey design objects handle all these details automatically, so you can ignore this problem.

The package test suite (`tests/domain.R`) verifies that subpopulation means agree with derivations from ratio estimators and regression estimator derivations. Some more documentation is in the domain vignette.

Note: subsets of design objects don't necessarily use less memory than the whole objects.

Quantiles

Confidence intervals use Woodruff's method by default

- Estimate the median \hat{m}
- Treat \hat{m} as fixed and get a confidence interval for $P(X > \hat{m})$
- Estimate the quantiles corresponding to the endpoints of the interval. This is the confidence interval

Quantiles: details

Two options for interpolation in the presence of ties

- **discrete**: the CDF has vertical steps at tied observations, interpolates linearly between distinct values
- **rounded**: the CDF is continuous, tied observations are treated as a single observation with the total weight

Default confidence interval is Normal-based, option `interval="betaWald"` uses the binomial distribution with a design effect (Korn & Graubard 1998, Shah & Vaish 2006).

Standard error is estimated by dividing the confidence interval length by twice the Normal critical value (eg 2×1.96)

Prettier tables

Two main types

- totals or proportions cross-classified by multiple factors
- arbitrary statistics in subgroups

Computing over subgroups

`svyby` computes a statistic for subgroups specified by a set of factor variables:

```
> svyby(~api99, ~stype, dclus1, svymean)
stype statistics.api99 se.api99
E E 607.7917 22.81660
H H 595.7143 41.76400
M M 608.6000 32.56064
```

`~api99` is the variable to be analysed, `~stype` is the subgroup variable, `dclus1` is the design object, `svymean` is the statistic to compute.

Lots of options for eg what variance summaries to present

Computing over subgroups

```
> svyby(~api99, ~stype, dclus1, svyquantile, quantiles=0.5, ci=TRUE)
```

stype	statistics.quantiles	statistics.CIs	se	var
E	E	615 525.6174, 674.1479	37.89113	1435.738
H	H	593 428.4810, 701.0065	69.52309	4833.46
M	M	611 527.5797, 675.2395	37.66903	1418.955

```
> svyby(~api99, list(school.type=apiclus1$stype), dclus1, svymean)
```

school.type	statistics.api99	se.api99
E	E	607.7917 22.81660
H	H	595.7143 41.76400
M	M	608.6000 32.56064

```
> svyby(~api99+api00, ~stype, dclus1, svymean, deff=TRUE)
```

```
> svyby(~api99+api00, ~stype, dclus1, svymean, deff=TRUE)
```

stype	statistics.api99	statistics.api00	se.api99	se.api00
E	E	607.7917	648.8681 22.81660	22.36241
H	H	595.7143	618.5714 41.76400	38.02025
M	M	608.6000	631.4400 32.56064	31.60947

	DEff.api99	DEff.api00
--	------------	------------

E	5.895734	6.583674
H	2.211866	2.228259
M	2.226990	2.163900

Computing over subgroups

	stype	sch.wide	statistic.api99	statistic.api00
E.No	E	No	601.6667	596.3333
H.No	H	No	662.0000	659.3333
M.No	M	No	611.3750	606.3750
E.Yes	E	Yes	608.3485	653.6439
H.Yes	H	Yes	577.6364	607.4545
M.Yes	M	Yes	607.2941	643.2353

Computing over subgroups

```
> (a<-svyby(~enroll, ~stype, rclus1, svytotal, deff=TRUE,
+ vartype=c("se","cv","cvpct","var")))
  stype statistics.enroll      se cv.enroll cv%.enroll      var      DEff
E     E      2109717.1 631349.4 0.2992578   29.92578 398602047550 125.039075
H     H       535594.9 226716.6 0.4232987   42.32987  51400414315   4.645816
M     M       759628.1 213635.5 0.2812369   28.12369  45640120138  13.014932
```

```
> deff(a)
[1] 125.039075 4.645816 13.014932
```

```
> SE(a)
[1] 631349.4 226716.6 213635.5
```

```
> cv(a)
[1] 0.2992578 0.4232987 0.2812369
```

```
> coef(a)
[1] 2109717.1 535594.9 759628.1
```

```
> svyby(~api00,~comp.imp+sch.wide,design=dclus1,svymean,
+ drop.empty.groups=FALSE)
```

```
      comp.imp sch.wide statistics.api00 se.api00
No.No      No      No      608.0435 28.98769
Yes.No     Yes      No           NA      NA
No.Yes     No      Yes      654.0741 32.66871
Yes.Yes    Yes      Yes      648.4060 22.47502
```

Functions of estimates

`svycontrast` computes linear and nonlinear combinations of estimated statistics (in the same object).

```
> a <- svytotal(~api00 + enroll + api99, dclus1)
> svycontrast(a, list(avg = c(0.5, 0, 0.5), diff = c(1,
0, -1)))
      contrast      SE
avg   3874804 873276
diff  230363  54921
> svycontrast(a, list(avg = c(api00 = 0.5, api99 = 0.5),
diff = c(api00 = 1, api99 = -1)))
      contrast      SE
avg   3874804 873276
diff  230363  54921
```

Functions of estimates

```
> svycontrast(a, quote(api00/api99))
              nlcon      SE
contrast 1.0613 0.0062
> svyratio(~api00, ~api99, dclus1)
Ratio estimator: svyratio.survey.design2(~api00, ~api99, dclus1)
Ratios=
              api99
api00 1.061273
SEs=
              api99
api00 0.006230831
```

The nonlinear transformation is useful for programming: was used to implement Cohen's kappa and the arcsin-sqrt transformation for proportions.

Crosstabs

`svyby` or `svymean` and `svytotal` with interaction will produce the numbers, but the formatting is not pretty.

`ftable` provides formatting:

```
> d<-svyby(~api99 + api00, ~stype + sch.wide, rclus1, svymean,  
           keep.var=TRUE, vartype=c("se","cvpct"))  
> round(ftable(d),1)
```

		No		Yes	
		statistics.api99	statistics.api00	statistics.api99	statistics.api00
stype					
E	svymean	601.7	596.3	608.3	653.6
	SE	70.0	64.5	23.7	22.4
	cv%	11.6	10.8	3.9	3.4
H	svymean	662.0	659.3	577.6	607.5
	SE	40.9	37.8	57.4	54.0
	cv%	6.2	5.7	9.9	8.9
M	svymean	611.4	606.4	607.3	643.2
	SE	48.2	48.3	49.5	49.3
	cv%	7.9	8.0	8.2	7.7

Crosstabs

`svyby` knows enough to structure the table without help. For other analyses more information is needed

```
> a<-svymean(~interaction(stype,comp.imp), design=dclus1, deff=TRUE)
> b<-ftable(a, rownames=list(stype=c("E","H","M"),comp.imp=c("No","Yes")))
> round(100*b,1)
```

	styp	E	H	M
comp.imp				
No	mean	17.5	3.8	6.0
	SE	2.6	1.6	2.5
	Deff	87.8	131.7	200.4
Yes	mean	61.2	3.8	7.7
	SE	4.2	1.6	2.2
	Deff	137.2	131.4	124.7

Testing in tables

`svychisq` does four variations on the Pearson χ^2 test: corrections to the mean or mean and variance of X^2 (Rao and Scott) and two Wald-type tests (Koch et al).

The exact asymptotic distribution of the Rao–Scott tests (linear combination of χ_1^2) is also available.

Loglinear models

`svyloglin()` does loglinear models

```
a<-svyloglin(~backpain+neckpain+sex+sickleave, nhis)
a2<-update(a, ~.^2)
a3<-update(a, ~.^3)
b1<-update(a, ~.+(backpain*neckpain*sex)+sex*sickleave)
b2<-update(a, ~.+(backpain+neckpain)*sex+sex*sickleave)
b3<-update(a, ~.+(backpain:neckpain+sex:backpain+sex:neckpain
+sex:sickleave)
```

`anova()` method computes Rao–Scott working loglikelihood and working score tests, with the two Rao–Scott approximations for the p -value and the exact asymptotic distribution.

Loglinear models

```
> anova(a,a2)
Analysis of Deviance Table
Model 1: y ~ backpain + neckpain + sex + sickleave
Model 2: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
      backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
      sex:sickleave
Deviance= 3563.795 p= 0
Score= 4095.913 p= 0
> anova(a2,a3)
Analysis of Deviance Table
Model 1: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
      backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
      sex:sickleave
Model 2: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
      backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
      sex:sickleave + backpain:neckpain:sex + backpain:neckpain:sickleave +
      backpain:sex:sickleave + neckpain:sex:sickleave
Deviance= 11.55851 p= 0.02115692
Score= 11.58258 p= 0.02094331
> print(anova(a2,a3),pval="saddlepoint")
[...snip...]
Deviance= 11.55851 p= 0.02065965
Score= 11.58258 p= 0.02044939
```


Graphics

When complex sampling designs are analyzed with regression models it is more important to have good exploratory analysis methods.

Problems with survey data

- large data
- unequal weights.

Estimating populations

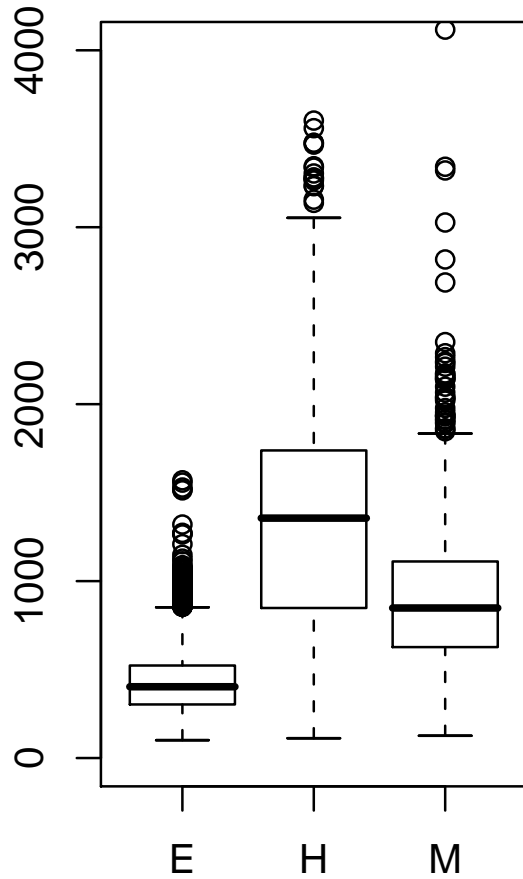
Boxplots, barplots, and histograms display estimates of the population distribution function for a variable.

We can substitute the survey estimates: boxplots use quantiles, histograms and barplots need tables. eg boxplot of enrollment by school type (Elementary/Middle/High)

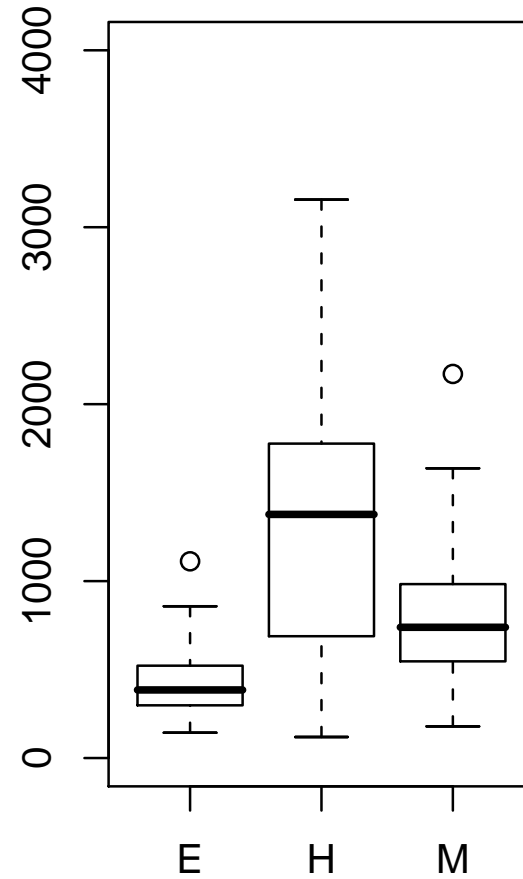
```
svyboxplot(enroll~stype, design=srs)
```

Estimating populations

Population



Sample



R does maps

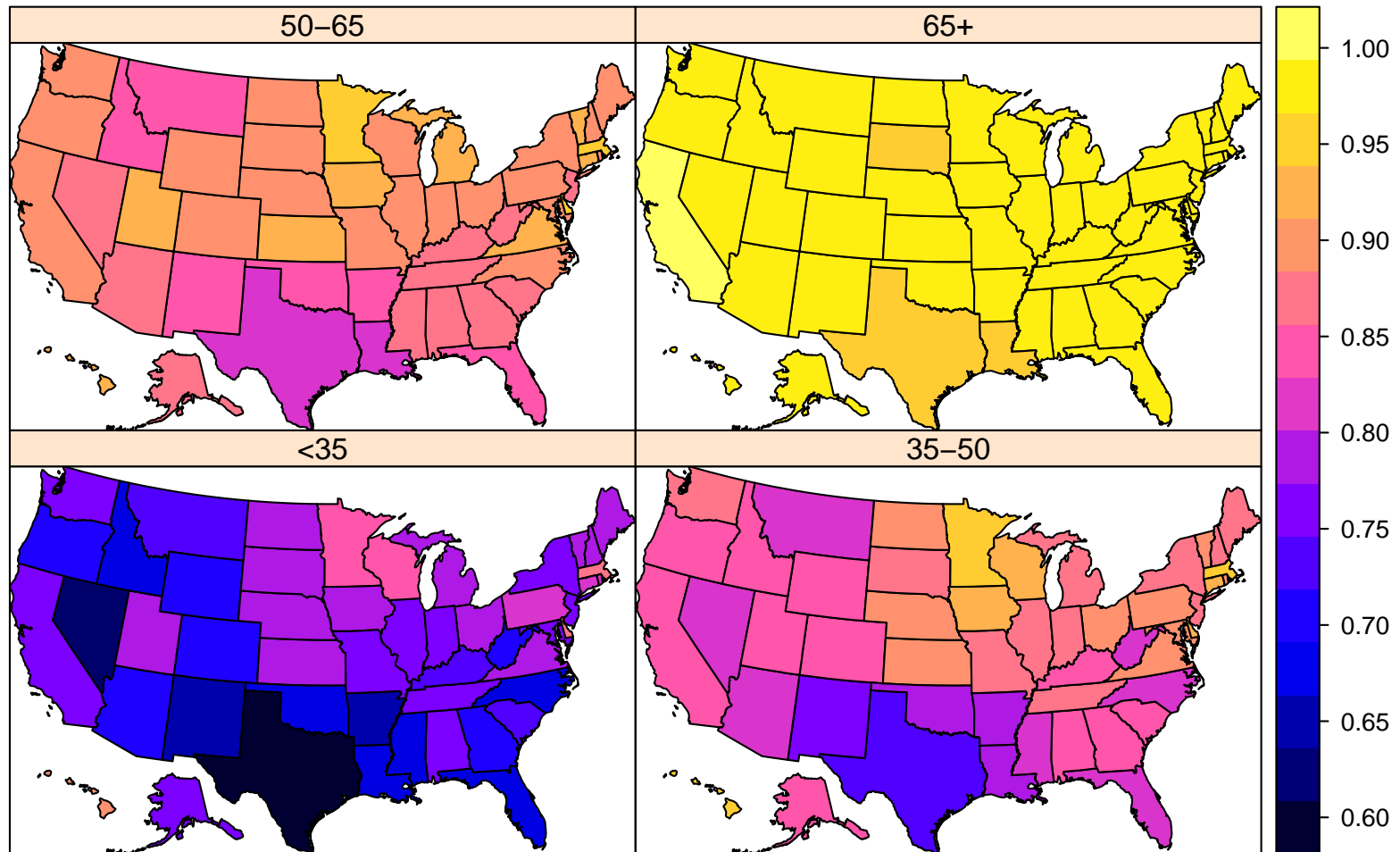
`maptools` and `sp` packages handle geographical data (eg ArcGIS shapefiles)

Estimate regional summaries with `svyby()`, merge output with GIS data, and then draw a map.

Example: health insurance by age and state, from BRFSS 2007.

GIS data for state outlines also come from BRFSS web site.

Health insurance coverage



Health insurance coverage

```
library(RSQLite)
brfss <- svydesign(id=~X_PSU, strata=~X_STATE, weight=~X_FINALWT,
  data="brfss", dbtype="SQLite", dbname="brfss07.db", nest=TRUE)
brfss<-update(brfss, agegp=cut(AGE, c(0,35,50,65,Inf)))

hlth<-svyby(~I(HLTHPLAN==1), ~agegp+X_STATE, svymean, design=brfss)
hlthdata<-reshape(hlth[,c(1,2,4)], idvar="X_STATE",
  direction="wide", timevar="agegp")
names(hlthdata)[2:5]<-paste("age",1:4,sep="")

states@data<-merge(states,hlthdata, by.x="ST_FIPS",by.y="X_STATE",
  all=FALSE)
spplot(states,c("age1","age2","age3","age4"),
  names.attr=c("<35","35-50","50-65","65+"))
```

Scatterplots

This approach doesn't work for scatterplots. We need to incorporate weights in the plotting symbols.

One approach is the bubble plot: radius or area of circle proportional to sampling weight.

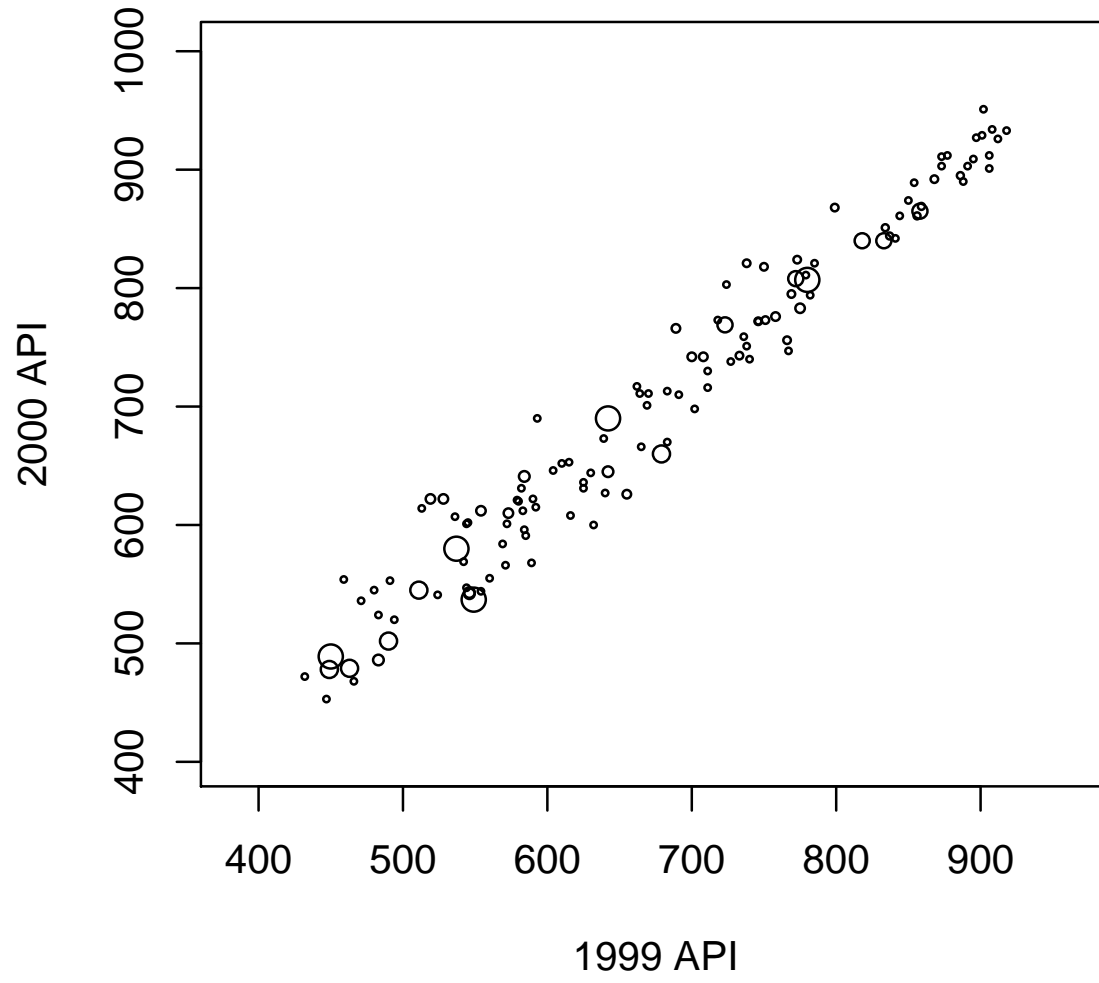
Another is transparent color: opacity for each point is proportional to sampling weight, giving a density estimation effect.

Bubble plots work well with small data sets, badly with large data sets.

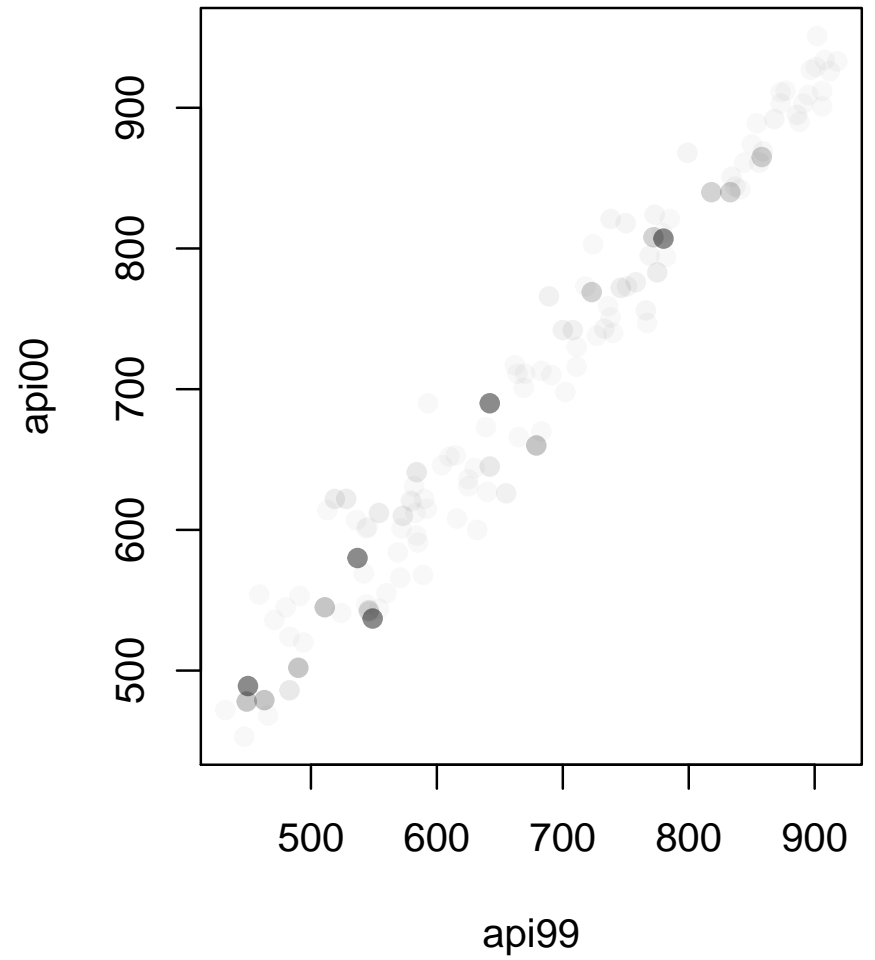
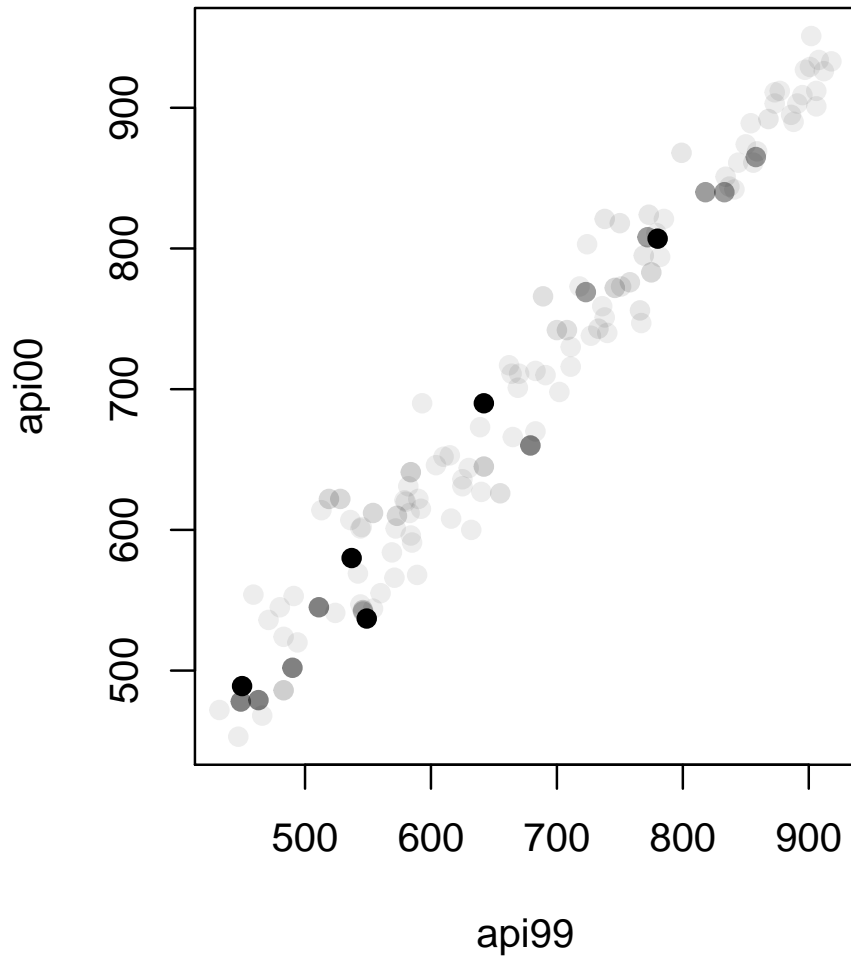
Transparency works well with large data sets, badly with small data sets.

Transparency allows color.

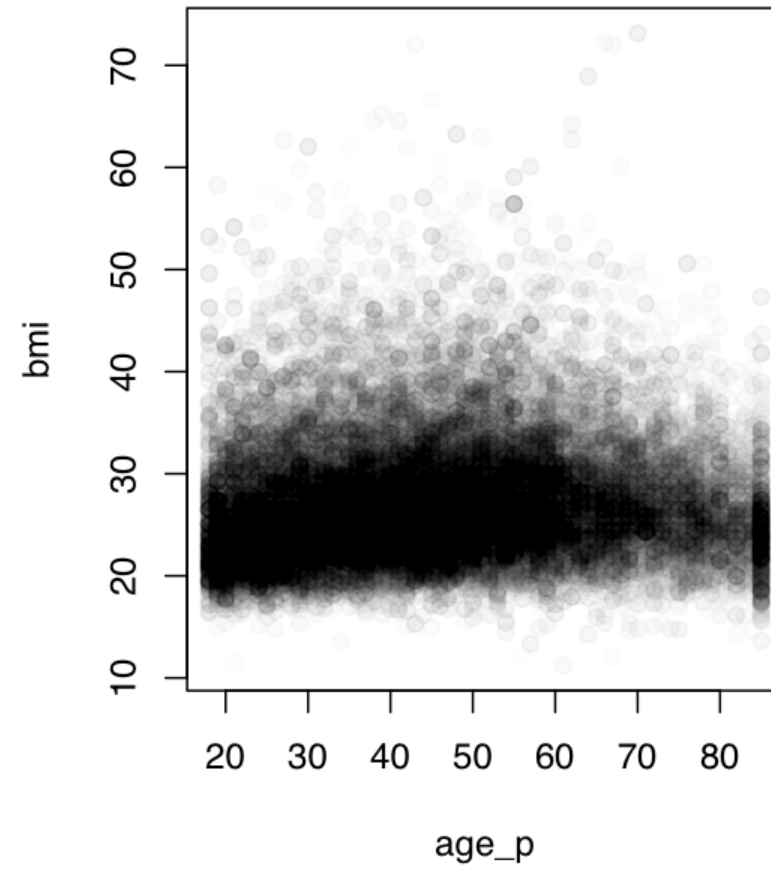
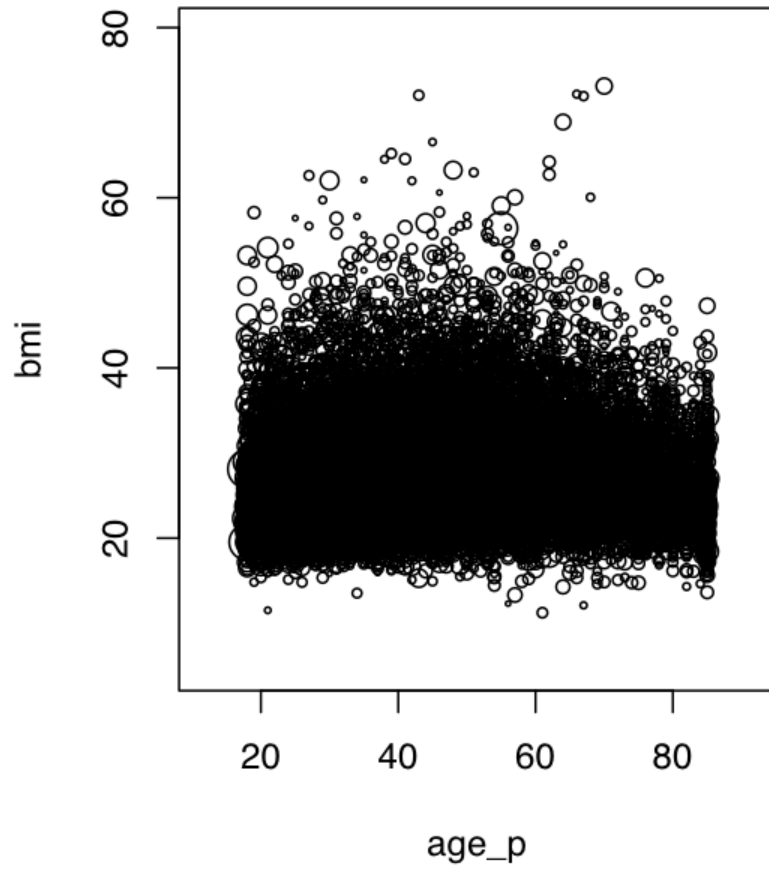
Scatterplots



Scatterplots



Scatterplots



Binning and smoothing

Hexagonal binning divides the plotting area into hexagons, counts the number in each hexagon [Carr et al, 1987, JASA]. In dense regions this has similar effect to transparency, but allows outliers to be seen, and produces much smaller graphics files.

For survey data we just add up the weight in each bin to get the estimated population numbers in each bin.

In R: `svyplot(bmi~age_p, design=nhis, style="hex")`

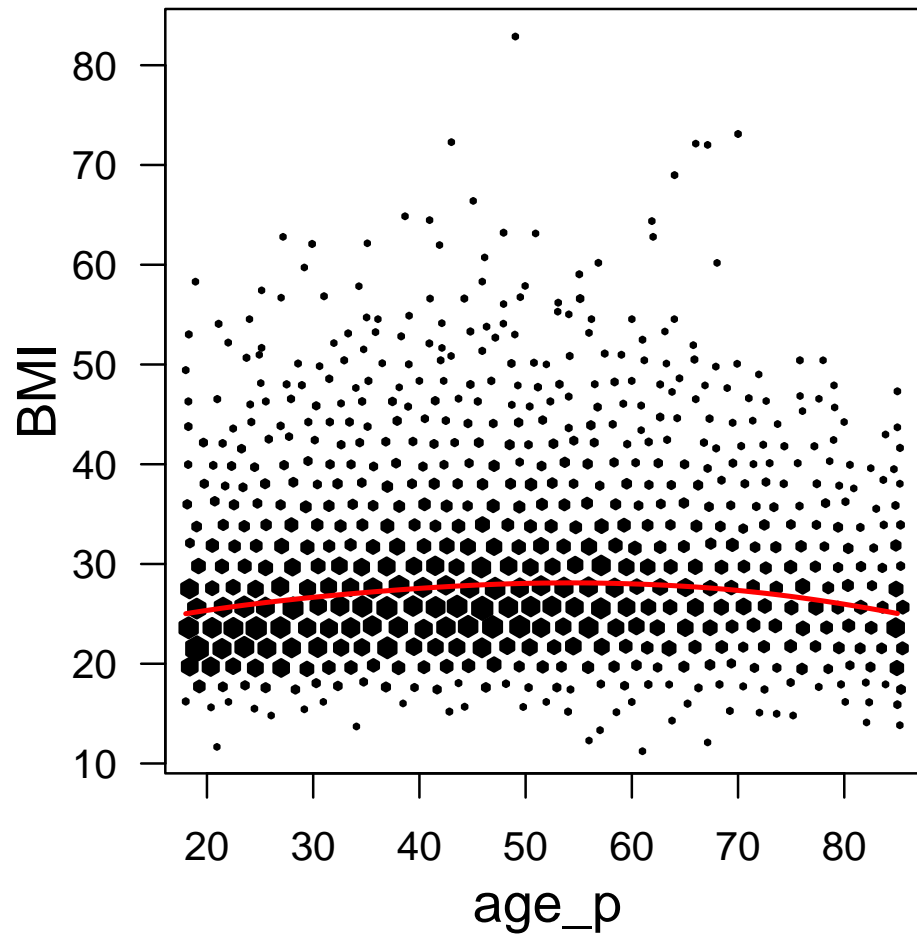
Binning and smoothing

Scatterplot smoothers fit a smooth curve through the middle of the data. There are many variants, but they all work by estimating the mean value of Y using points in a small interval of values of x . `svysmooth()` has a binned kernel smoother for the mean and a quantile regression spline smoother for quantiles.

In R use `svysmooth` to create a smooth curve and then `plot` to draw it.

```
smth <- svysmooth(api00~api99, strattrs, bandwidth=40)
plot(smth)
```

Binning and smoothing



Counts

- 2265009
- 2123446
- 1981883
- 1840320
- 1698757
- 1557194
- 1415631
- 1274068
- 1132505
- 990942
- 849379
- 707816
- 566253
- 424690
- 283127
- 141564
- 1

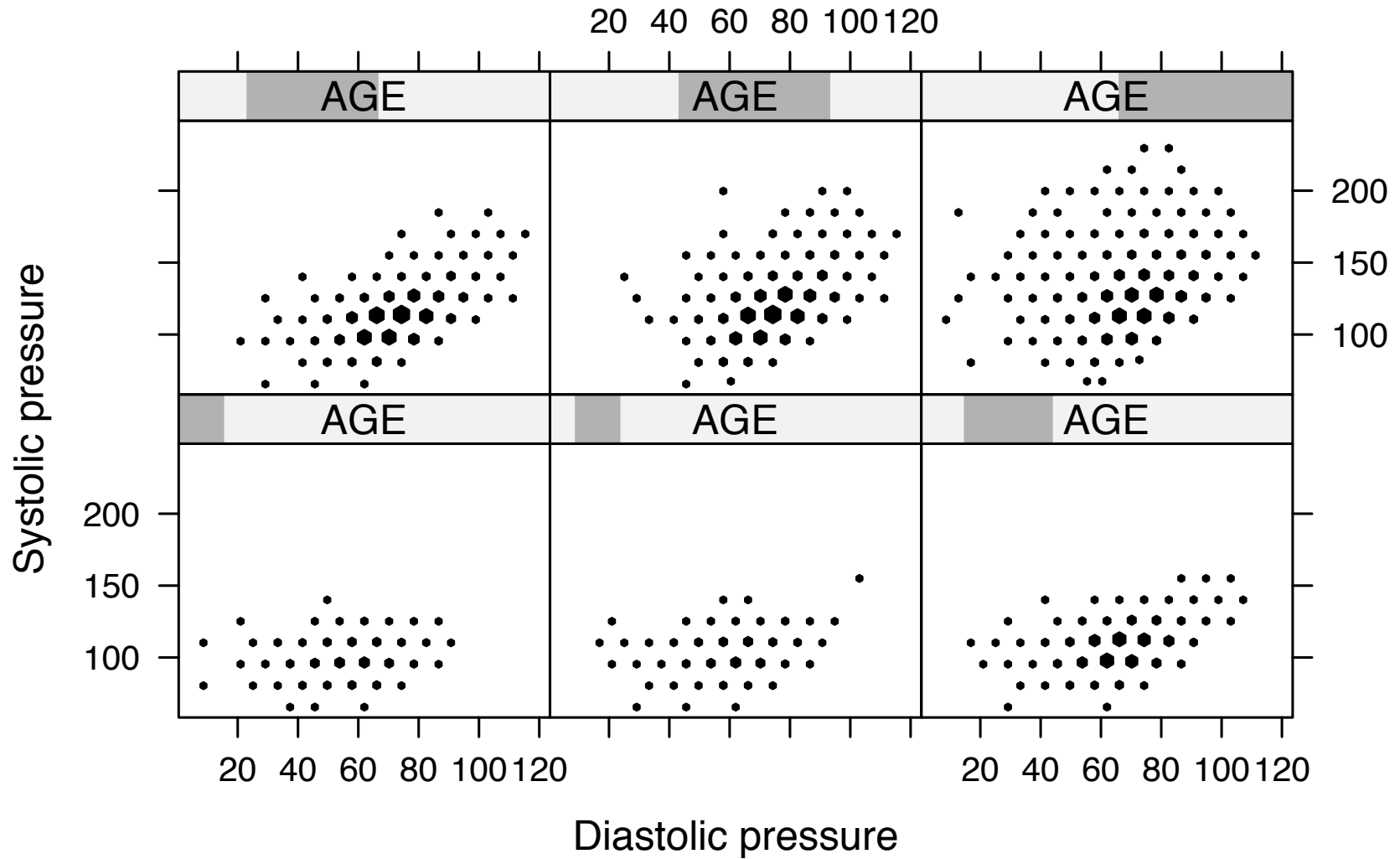
Conditioning plots

`svycoplot()` does conditioning plots, which can either use hexagonal binning or transparency

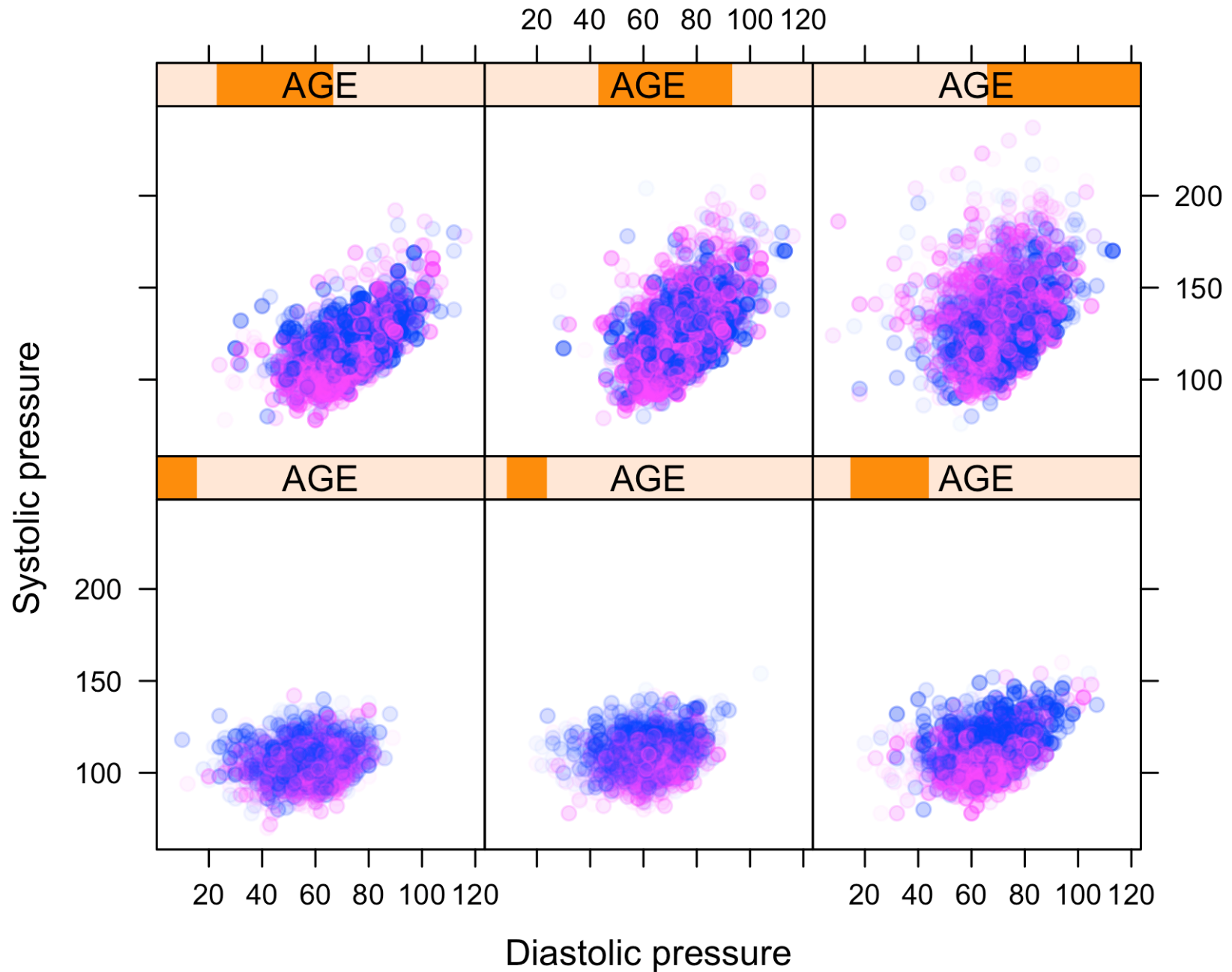
```
svycoplot(BPXSAR~BPXDAR|equal.count(RIDAGEMN), style="hex",  
  design=subset(dhanes,BPXDAR>0), xbins=20,  
  strip=strip.custom(var.name="AGE"),  
  xlab="Diastolic pressure",ylab="Systolic pressure")
```

```
svycoplot(BPXSAR~BPXDAR|equal.count(RIDAGEMN), style="trans",  
  design=subset(dhanes,BPXDAR>0),  
  strip=strip.custom(var.name="AGE"),  
  xlab="Diastolic pressure",ylab="Systolic pressure",  
  basecol=function(d) ifelse(d$RIAGENDR==2,"magenta","blue"))
```

Conditioning plots



Conditioning plots



Multivariate data

Biplots of principal components are useful for summarizing multivariate data

- (population) principal components estimated from singular value decomposition of weighted data matrix
- `biplot()` shows points and singular vectors for the first two components, with sampling weights indicated by transparency or scale

```
> data(api)
> dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
> pc <- svyprcomp(~api99+api00+ell+hsg+meals+emer, design=dclus2,
  scale=TRUE,scores=TRUE)
> pc
```

Multivariate data

Standard deviations:

```
[1] 2.006147 1.067846 0.728197 0.515939 0.188624 0.055115
```

Rotation:

	PC1	PC2	PC3	PC4	PC5	PC6
api99	0.38311	-0.59436	0.083068	-0.069799	0.0223023	-0.6983489
api00	0.39349	-0.57188	0.038367	-0.077049	-0.0082088	0.7145931
ell	0.44937	0.20566	-0.256407	0.587637	0.5871150	0.0010088
hsg	0.40895	0.35068	-0.213175	-0.776285	0.2480553	-0.0139542
meals	0.45944	0.23847	-0.321126	0.192964	-0.7685087	-0.0329332
emer	0.34369	0.31234	0.881656	0.063414	-0.0509340	0.0196242

Regression models

- `svyglm` for linear and generalized linear models (and regression estimator of total via `predict()`)
- `svyolr` for proportional odds and other cumulative link models.
- `svycoxph` for Cox model

For these models the point estimates are the same for frequency weights, sampling weights, or precision weights, so the point estimation can just reuse the ordinary regression code.

Regression model

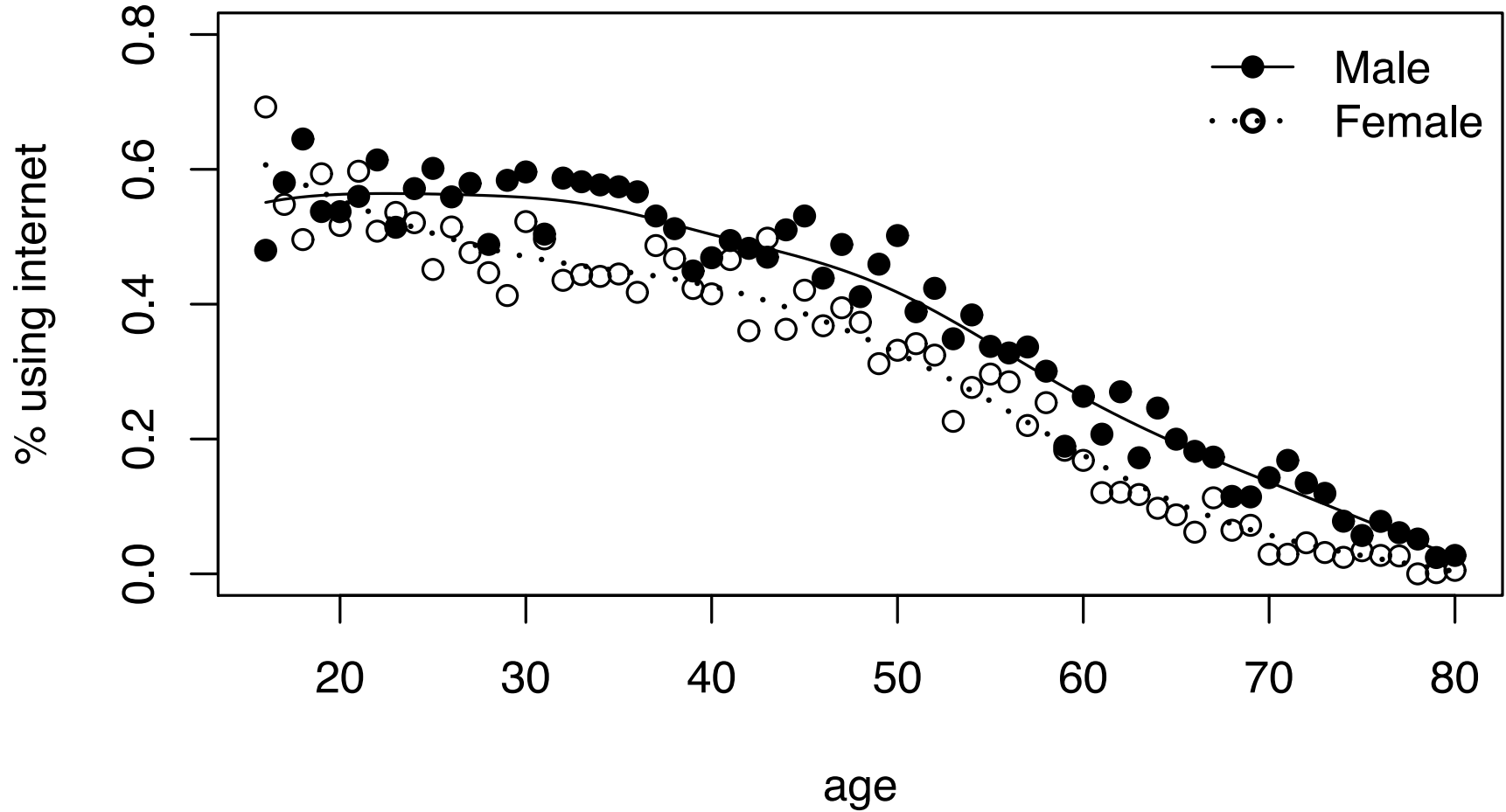
Internet use in Scotland (2001) by age, sex, and income.

```
shs<-svydesign(id=~psu, strata=~stratum, weight=~grosswt,
             data=shs_data)

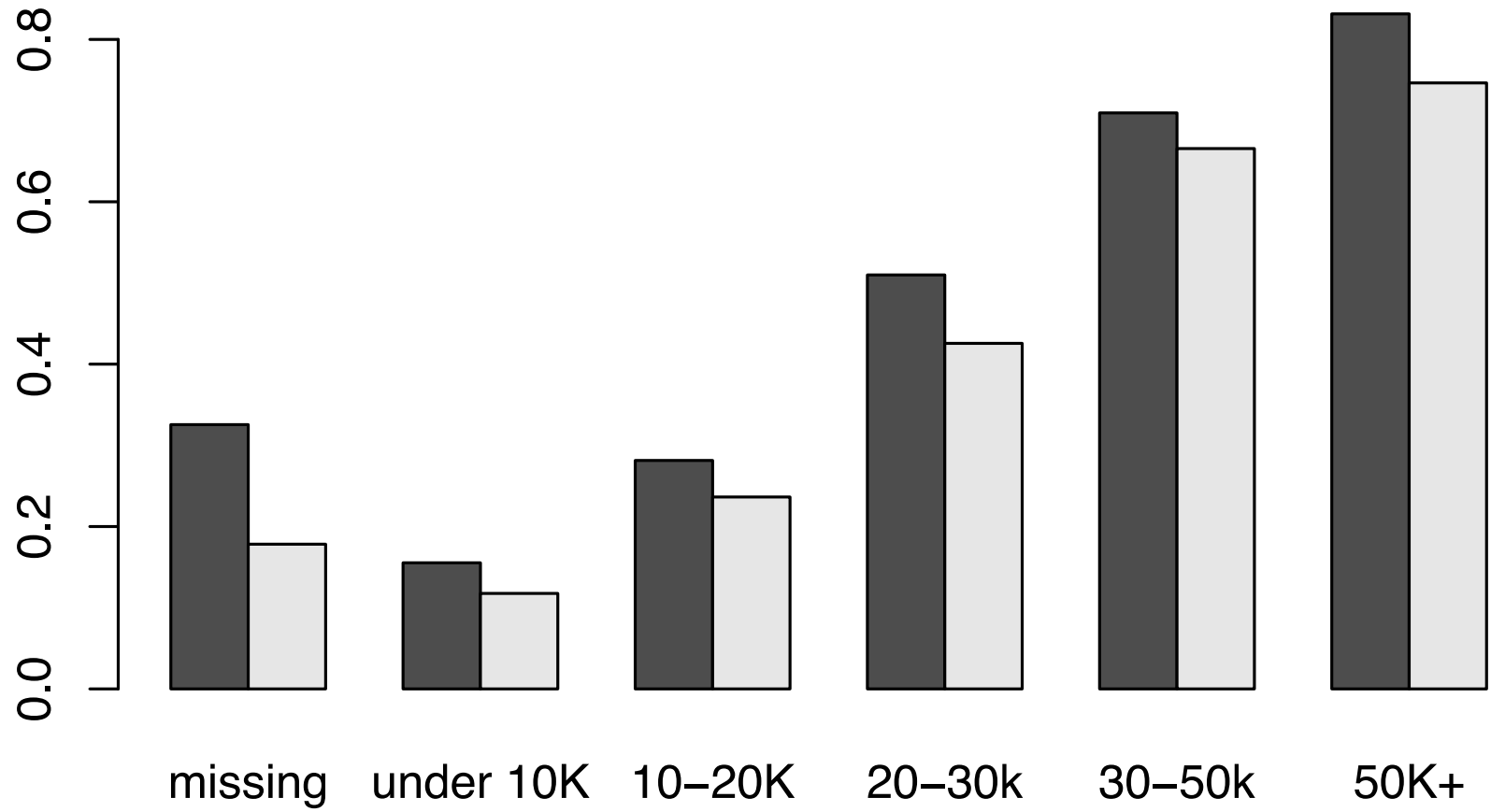
plot(svysmooth(intuse~age,
              design=subset(shs,sex=="male" & !is.na(age)),
              bandwidth=5),ylim=c(0,0.8),ylab="% using internet")
lines(svysmooth(intuse~age,
              design=subset(shs,sex=="female" & !is.na(age)),
              bandwidth=5),lwd=2,lty=3)
points(bys$age,bys$intuse,pch=ifelse(bys$sex=="male",19,1))
legend("topright",pch=c(19,1),lty=c(1,3),lwd=c(1,2),
      legend=c("Male","Female"),bty="n")

byinc<-svyby(~intuse, ~sex+groupinc, design=shs)
barplot(byinc)
```

Age (cohort) effect



Income



Code

```
> m<-svyglm(intuse~I(age-18)*sex,design=shs,
  family=quasibinomial())
> m2<-svyglm(intuse~(pmin(age,35)+pmax(age,35))*sex,
  design=shs,family=quasibinomial)
> summary(m)
svyglm(intuse ~ I(age - 18) * sex, design = shs,
  family = quasibinomial())
Survey design:
svydesign(id = ~psu, strata = ~stratum, weight = ~grosswt,
  data = ex2)
Coefficients:

```

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	0.804113	0.047571	16.903	< 2e-16	***
I(age - 18)	-0.044970	0.001382	-32.551	< 2e-16	***
sexfemale	-0.116442	0.061748	-1.886	0.0594	.
I(age - 18):sexfemale	-0.010145	0.001864	-5.444	5.33e-08	***

Code

```
> summary(m2)
```

```
Call:
```

```
svyglm(intuse ~ (pmin(age, 35) + pmax(age, 35)) * sex,  
       design = shs, family = quasibinomial)
```

```
Survey design:
```

```
svydesign(id = ~psu, strata = ~stratum, weight = ~grosswt,  
         data = ex2)
```

```
Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	2.152291	0.156772	13.729	< 2e-16	***
pmin(age, 35)	0.014055	0.005456	2.576	0.010003	*
pmax(age, 35)	-0.063366	0.001925	-32.922	< 2e-16	***
sexfemale	0.606718	0.211516	2.868	0.004133	**
pmin(age, 35):sexfemale	-0.017155	0.007294	-2.352	0.018691	*
pmax(age, 35):sexfemale	-0.009804	0.002587	-3.790	0.000151	***

Code

```
> svycontrast(m2,
  quote('pmin(age, 35)' + 'pmin(age, 35):sexfemale'))
      nlcon      SE
contrast -0.0031 0.0049
> svycontrast(m2,
  quote('pmax(age, 35)' + 'pmax(age, 35):sexfemale'))
      nlcon      SE
contrast -0.07317 0.0018
```

Simulations

One advantage of having survey analysis tools in a statistical programming environment is that simulations are easy.

Example: comparing quantile confidence intervals:

- take cluster sample of school districts from California
- estimate quantiles of `e11` and confidence intervals
- repeat.

Simulations

```
apipop$ndnum <- 757
one.sim <- function(p=0.1, m=15){
  districts <- sample(unique(apipop$dnum), m)
  design <- svydesign(id=~dnum, fpc=~ndnum,
    data=subset(apipop, dnum %in% districts))
  svyquantile(~ell, design, quantiles=p, ci=TRUE)$CIs
}
```

Simulations

Test the code

```
> one.sim()
```

```
, , ell
```

```
0.1
```

```
(lower 0
```

```
upper) 2
```

```
> one.sim(.5,400)
```

```
, , ell
```

```
0.5
```

```
(lower 12
```

```
upper) 16
```

```
> system.time(replicate(3, one.sim()))
```

```
user system elapsed
```

```
0.089 0.017 0.110
```

Simulations

```
> system.time(results<-replicate(1000, one.sim(.1)))
  user  system elapsed
30.262   5.926  36.320
> str(results)
 num [1:2, 1:1000] 0 4 2.12 12 0.851 ...

> mean(results[1,] > 1)
[1] 0.021
> mean(results[2,] < 1)
[1] 0.002
> mean(results[1,] >= 1)
[1] 0.063
> mean(results[2,] <= 1)
[1] 0.081
```

Simulations

Now repeat for `interval.type="betaWald"`

```
> mean(results1[1,] > 1)
[1] 0.034
> mean(results1[2,] < 1)
[1] 0
> mean(results1[1,] >= 1)
[1] 0.099
> mean(results1[2,] <= 1)
[1] 0.008
```

Slightly better coverage: less conservatism

```
## now try other quantiles
results.all<-lapply(c(0.1,0.25,0.5, 0.75,0.9),
                    function(p) replicate(1000, one.sim(p)))
```

Coverage is much worse at higher quantiles.

Post-stratification and calibration

Post-stratification and calibration are ways to use auxiliary information on the population (or the phase-one sample) to improve precision.

They are closely related to the Augmented Inverse-Probability Weighted estimators of Jamie Robins and coworkers, but are easier to understand.

Auxiliary information

HT estimator is inefficient when some additional population data are available.

Suppose x_i is known for all i

Fit $y \sim x\beta$ by (probability-weighted) least squares to get $\hat{\beta}$. Let r^2 be proportion of variation explained.

$$\hat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i} (y_i - x_i \hat{\beta}) + \sum_{i=1}^N x_i \hat{\beta}$$

ie, HT estimator for sum of residuals, plus population sum of fitted values

Auxiliary information

Let β^* be true value of β (ie, least-squares fit to whole population).

Regression estimator

$$\hat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i} (y_i - x_i \beta^*) + \left(\sum_{i=1}^N x_i \right) \beta^* + \sum_{i=1}^N \left(1 - \frac{R_i}{\pi_i} \right) x_i (\hat{\beta} - \beta^*)$$

compare to HT estimator

$$\hat{T} = \sum_{R_i=1} \frac{1}{\pi_i} (y_i - x_i \beta^*) + \left(\sum_{R_i=1} \frac{1}{\pi_i} x_i \right) \beta^*$$

Second term uses known vs observed total of x , third term is estimation error for β , of smaller order.

Auxiliary information

For large n , N and under conditions on moments and sampling schemes

$$\text{var} [\hat{T}_{reg}] = (1-r^2) \text{var} [\hat{T}] + O(N/\sqrt{n}) = (1 - r^2 + O(n^{-1/2})) \text{var} [\hat{T}]$$

and the relative bias is $O(1/n)$

The lack of bias does not require any assumptions about $[Y|X]$

$\hat{\beta}$ is consistent for the population least squares slope β , for which the mean residual is zero by construction.

Reweighting

Since $\hat{\beta}$ is linear in y , we can write $x\hat{\beta}$ as a linear function of y and so \hat{T}_{reg} is also a linear function of Y

$$\hat{T}_{reg} = \sum_{R_i=1} w_i y_i = \sum_{R_i=1} \frac{g_i}{\pi_i} y_i$$

for some (ugly) w_i or g_i that depend only on the x s

For these weights

$$\sum_{i=1}^N x_i = \sum_{R_i=1} \frac{g_i}{\pi_i} x_i$$

\hat{T}_{reg} is an IPW estimator using weights that are ‘calibrated’ or ‘tuned’ (French: *calage*) so that the known population totals are estimated correctly.

Calibration

The general calibration problem: given a distance function $d(\cdot, \cdot)$, find **calibration weights** g_i minimizing

$$\sum_{R_i=1} d(g_i, 1)$$

subject to the **calibration constraints**

$$\sum_{i=1}^N x_i = \sum_{R_i=1} \frac{g_i}{\pi_i} x_i$$

Lagrange multiplier argument shows that $g_i = \eta(x_i\beta)$ for some $\eta(\cdot)$, β ; and γ can be computed by iteratively reweighted least squares.

For example, can choose $d(\cdot, \cdot)$ so that g_i are bounded below (and above).

[Deville *et al* JASA 1993; JNK Rao *et al*, Sankhya 2002]

Calibration

When the calibration model in x is saturated, the choice of $d(,)$ does not matter: calibration equates estimated and known category counts.

In this case calibration is also the same as estimating sampling probabilities with logistic regression, which also equates estimated and known counts.

Calibration to a saturated model gives the same analysis as pretending the sampling was stratified on these categories: **post-stratification**

Post-stratification is a much older method, and is computationally simpler, but calibration can make more use of auxiliary data.

Standard errors

Standard errors come from the regression formulation

$$\hat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i} (y_i - x_i \hat{\beta}) + \sum_{i=1}^N x_i \hat{\beta}$$

The variance of the second term is of smaller order and is ignored.

The variance of the first term is the usual Horvitz–Thompson variance estimator, applied to residuals from projecting y on the calibration variables.

$$\widehat{\text{var}} = \sum_{i,j=1}^n \check{\Delta}_{ij} g_i \check{r}_i g_j \check{r}_j$$

Computing

R provides `calibrate()` for calibration (and `postStratify()` for post-stratification)

Three basic types of calibration

- Linear (or regression) calibration: identical to regression estimator
- Raking: multiplicative model for weights, guarantees $g_i > 0$
- Logit calibration: logit link for weights, popular in Europe, provides upper and lower bounds for g_i

Computing

Upper and lower bounds for g_i can also be specified for linear and raking calibration (these may not be achievable, but we try).

There is also an option to trim weights to specified bounds, even if these do not achieve the calibration constraints.

The user can specify other calibration loss functions (eg Hellinger distance).

Computing

The `calibrate()` function takes three main arguments

- a survey design object
- a model formula describing the design matrix of auxiliary variables
- a vector giving the column sums of this design matrix in the population.

and additional arguments describing the type of calibration.

Computing

```
> data(api)
> dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
> pop.totals<-c('(Intercept)'=6194, stypeH=755, stypeM=1018)

> (dclus1g<-calibrate(dclus1, ~stype, pop.totals))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype, pop.totals)
> svymean(~api00, dclus1g)
      mean      SE
api00 642.31 23.921
> svymean(~api00,dclus1)
      mean      SE
api00 644.17 23.542
```

Computing

```
> svytotal(~enroll, dclus1g)
```

```
      total      SE
enroll 3680893 406293
```

```
> svytotal(~enroll, dclus1)
```

```
      total      SE
enroll 3404940 932235
```

```
> svytotal(~stype, dclus1g)
```

```
      total      SE
stypeE  4421 1.118e-12
stypeH   755 4.992e-13
stypeM  1018 1.193e-13
```

Computing

```
> (dclus1g3 <- calibrate(dclus1, ~stype+api99,
                        c(pop.totals, api99=3914069)))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069))
> svymean(~api00, dclus1g3)
      mean      SE
api00 665.31 3.4418
> svyttotal(~enroll, dclus1g3)
      total      SE
enroll 3638487 385524
> svyttotal(~stype, dclus1g3)
      total      SE
stypeE  4421 1.179e-12
stypeH   755 4.504e-13
stypeM  1018 9.998e-14
```

Computing

```
> range(weights(dclus1g3)/weights(dclus1))
```

```
[1] 0.4185925 1.8332949
```

```
> (dclus1g3b <- calibrate(dclus1, ~stype+api99,  
      c(pop.totals, api99=3914069), bounds=c(0.6,1.6)))
```

```
1 - level Cluster Sampling design
```

```
With (15) clusters.
```

```
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),  
      bounds = c(0.6, 1.6))
```

```
> range(weights(dclus1g3b)/weights(dclus1))
```

```
[1] 0.6 1.6
```

Computing

```
> svymean(~api00, dclus1g3b)
```

	mean	SE
api00	665.48	3.4184

```
> svyttotal(~enroll, dclus1g3b)
```

	total	SE
enroll	3662213	378691

```
> svyttotal(~stype, dclus1g3b)
```

	total	SE
stypeE	4421	1.346e-12
stypeH	755	4.139e-13
stypeM	1018	8.238e-14

Computing

```
> (dclus1g3c <- calibrate(dclus1, ~stype+api99, c(pop.totals,
+      api99=3914069), calfun="raking"))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),
  calfun = "raking")
> range(weights(dclus1g3c)/weights(dclus1))
[1] 0.5342314 1.9947612
> svymean(~api00, dclus1g3c)
      mean      SE
api00 665.39 3.4378
```


Computing

```
> (dclus1g3d <- calibrate(dclus1, ~stype+api99, c(pop.totals,
+      api99=3914069), calfun="logit", bounds=c(0.5,2.5)))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),
  calfun = "logit", bounds = c(0.5, 2.5))
> range(weights(dclus1g3d)/weights(dclus1))
[1] 0.5943692 1.9358791
> svymean(~api00, dclus1g3d)
      mean      SE
api00 665.43 3.4325
```

Types of calibration

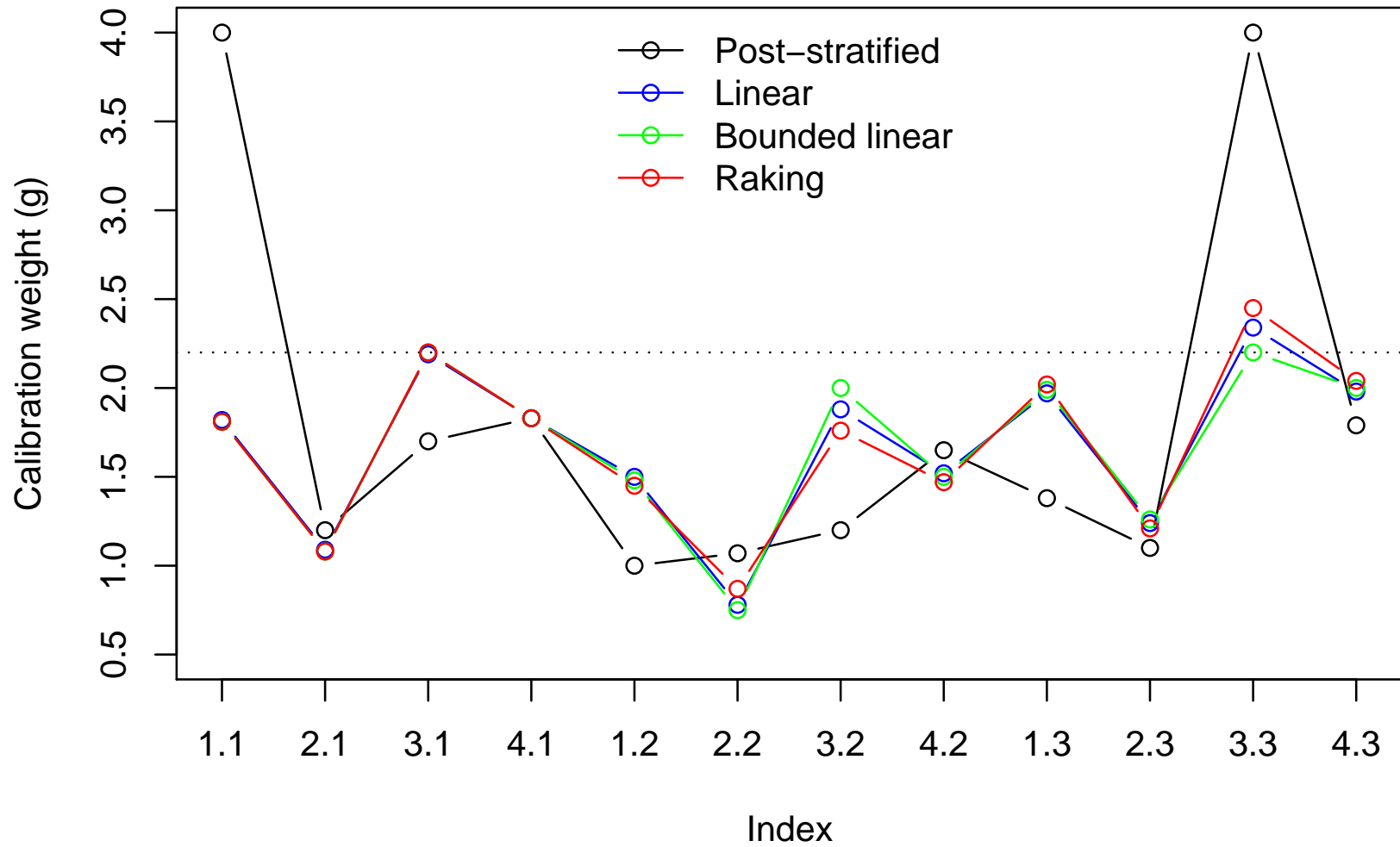
Post-stratification allows much more flexibility in weights, in small samples can result in very influential points, loss of efficiency.

Calibration allows for less flexibility (cf stratification vs regression for confounding)

Different calibration methods make less difference

Example from Kalton & Flores-Cervantes (J. Off. Stat, 2003):
a 3×4 table of values.

Types of calibration



Using multiply-imputed data

Multiple imputation of missing data: fit a model to the data, simulate multiple possibilities for the missing data from the predictive distribution of the model. (Rubin, 1978)

Do the same analysis to each completed data set

Point estimate is the average of the point estimates

Variance estimate is the average variance + variance between point estimates.

Simple approach is technically valid only for 'proper' imputations including posterior uncertainty in model parameters, which is inefficient. [Wang & Robins, Bka 1998]

Using multiply-imputed data

Need code to do repeated analysis, combine results.

- `imputationList()` wraps a list of data frames or database tables
- `svydesign()` can take an `imputationList` as the data argument to give a set of designs.
- `with(designs, expr)` runs `expr`, with `design=` each one of the designs in turn
- `MIcombine()` combines the results.

NHANES III imputations

```
> library(mitools)
> library(RSQLite)
> impdata <- imputationList(c("set1", "set2", "set3", "set4", "set5"),
                           dbtype="SQLite", dbname="~/nhanes/imp.db")
> impdata
MI data with 5 datasets
Call: imputationList(c("set1", "set2", "set3", "set4", "set5"),
                     dbtype = "SQLite", dbname = "~/nhanes/imp.db")
> designs <- svydesign(id=~SDPPSU6, strat=~SDPSTRA6,
                     weight=~WTPFQX6, data=impdata, nest=TRUE)
> designs
DB-backed Multiple (5) imputations: svydesign(id = ~SDPPSU6,
        strat = ~SDPSTRA6, weight = ~WTPFQX6,
        data = impdata, nest = TRUE)
```

NHANES III imputations

```
> designs<-update(designs,  
  age=ifelse(HSAGEU==1, HSAGEIR/12, HSAGEIR))  
> designs<-update(designs,  
  agegp=cut(age,c(20,40,60,Inf),right=FALSE))  
> res <- with(subset(designs, age>=20),  
  svyby(~BDPFNDMI, ~agegp+HSSEX, svymean))  
> summary(MIcombine(res))
```

Multiple imputation results:

```
with(subset(designs, age >= 20), svyby(~BDPFNDMI,  
  ~agegp + HSSEX, svymean, design = .design))  
MIcombine.default(res)
```

	results	se	(lower	upper)	missInfo
[20,40).1	0.9355049	0.003791945	0.9279172	0.9430926	28 %
[40,60).1	0.8400738	0.003813224	0.8325802	0.8475674	10 %
[60,Inf).1	0.7679224	0.004134875	0.7598032	0.7760416	8 %
[20,40).2	0.8531107	0.003158246	0.8468138	0.8594077	26 %
[40,60).2	0.7839377	0.003469386	0.7771144	0.7907610	11 %
[60,Inf).2	0.6454393	0.004117235	0.6370690	0.6538096	38 %

Two-phase subsampling

- Phase 1: sample people according to some probability design $\pi_{1,i}$, measure variables
- Phase 2: subsample people from the phase 1 sample using the variables measured at phase 1, $\pi_{2|1,i}$ and measure more variables

Sampling probability $\pi_i = \pi_{1,i} \times \pi_{2|1,i}$ and use $1/\pi_i$ as sampling weights.

These are **not** (in general) the marginal probability that i is in the sample, because $\pi_{2|1,i}$ may depend on which other observations are in the phase-one sample.

Two-phase subsampling

Two sources of uncertainty:

- Phase-1 sample is only part of population
- Phase 2 observes full set of variables on only a subset of people.

Uncertainties at the two phases add.

Minimal phase 1

The classic survey example is ‘two-phase sampling for stratification’. This is useful when a good stratifying variable is not available for the population but is easy to measure.

- Take a large simple random sample or cluster sample and measure stratum variables
- Take a stratified random sample from phase 1 for the survey

If the gain from stratification is larger than the cost of phase 1 we have won.

The case–control design is probably the most important example, although it usually isn’t analyzed using sampling weights.

Minimal phase 2

Many large cohorts exist in epidemiology. These can be modelled as simple random samples. They have a lot of variables measured.

It is common to want to measure a new variable

- New assay on stored blood
- Coding of open-text questionnaire
- Re-interview

The classic designs are a simple random sample and a case–control sample.

It is often more useful to sample based on multiple phase-1 variables: outcome, confounders, surrogates for phase-2 variable.

Math

We know the phase-one sampling probabilities $\pi_{i,1}$ and the probability of being sampled for phase 2 conditional on the actual phase-one sample, $\pi_{i,1|2}$.

We can use the reciprocal of $\pi_i^* = \pi_{i,1} \times \pi_{i,1|2}$ as a weight to get unbiased estimation.

The actual sampling probability for observation i is the average of π_i^* over all phase-1 samples including observation i , which is not feasible to compute, but the estimator based on π_i^* works just like the Horvitz–Thompson estimator.

Variance Algorithms

When phase 1 is simple random sampling there are shortcuts, but in general it seems easiest to use the general form

$$\widehat{\text{var}}[\hat{T}_x] = \sum_{i,j=1}^n \check{\Delta}_{ij}^* \check{x}_i \check{x}_j$$

where $\Delta^* = \pi_{ij}^* - \pi_i^* \pi_j^*$ and the check refers to scaling by the weights.

Often Δ^* is a sparse matrix, so this is not as inefficient as it looks and $\check{\Delta}^*$ is easy to compute from Δ at each phase.

Two-phase case-control

The case-control design stratifies on Y . We can stratify on X as well

	X=0	X=1	
Y=0	a	b	m_0
Y=1	c	d	m_1
	n_0	n_1	

The estimated variance of β is

$$\text{var}[\hat{\beta}] = \frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}$$

Ideally want all cells about the same size in this trivial case.

Example: Wilm's Tumor

- Wilm's Tumor is a rare childhood cancer of the kidney. Prognosis is good in early stage or favourable histology disease.
- Histology is difficult to determine. NWTSG central pathologist is much better than anyone else.
- To reduce cost of followup, consider central histology (`histol`) only for a subset of cases.
- Sample all relapses, all patients with unfavorable histology (`instit`) by local pathologist, 10% of remainder

Analyses: sampling weights

Phase 1

	relapse	
instit	0	1
good	3207	415
bad	250	156

Phase 2

	relapse	
instit	0	1
good	314	415
bad	250	156

Weight is $1/\text{sampling fraction in relapse} \times \text{local histology strata}$: 1.0 for cases, controls with unfavorable local histology, $3207/314 = 10.2$ for controls with favorable local histology.

Computation in R

Declaring a two-phase design

```
dccs2 <- twophase(id = list(~id, ~id), subset = ~in.ccs,  
                 strata = list(NULL, ~interaction(instit, rel)),  
                 data = nwt.exp)
```

- Data set has records for all phase-one people, `subset` variable indicates membership in second phase
- Two `id`, two `strata`.
- Second-phase `weights` and `fpc` are worked out by R

Computation in R

We can compare to the conservative approximation: unstratified single-phase sampling with replacement

```
dcons<-svydesign(id=~seqno, weights=weights(dccs2),  
               data=subset(nwtco, incc2))
```

Almost no advantage of two-phase analysis in this example; but wait until later with calibration of weights.

Computation in R

```
> summary(svyglm(rel~factor(stage)*factor(histol),design=dccs2,
                 family=quasibinomial()))
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      -2.6946    0.1246 -21.623  < 2e-16 ***
factor(stage)2     0.7733    0.1982   3.901 0.000102 ***
factor(stage)3     0.7400    0.2048   3.614 0.000315 ***
factor(stage)4     1.1322    0.2572   4.401 1.18e-05 ***
factor(histol)2    1.1651    0.3196   3.646 0.000279 ***
factor(stage)2:factor(histol)2  0.3642    0.4462   0.816 0.414511
factor(stage)3:factor(histol)2  1.0230    0.3968   2.578 0.010056 *
factor(stage)4:factor(histol)2  1.7444    0.4973   3.508 0.000470 ***
```

```
> summary(svyglm(rel~factor(stage)*factor(histol),design=dcons,
                 family=quasibinomial()))
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      -2.6946    0.1355 -19.886  < 2e-16 ***
factor(stage)2     0.7733    0.1982   3.902 0.000101 ***
factor(stage)3     0.7400    0.2047   3.615 0.000314 ***
factor(stage)4     1.1322    0.2571   4.403 1.17e-05 ***
factor(histol)2    1.1651    0.3208   3.632 0.000294 ***
factor(stage)2:factor(histol)2  0.3642    0.4461   0.816 0.414408
factor(stage)3:factor(histol)2  1.0230    0.3974   2.574 0.010169 *
factor(stage)4:factor(histol)2  1.7444    0.4977   3.505 0.000474 ***
```

Computation in R

We are not restricted to logistic regression: eg, log link gives log relative risks rather than log odds ratios. Since relapse is not rare for unfavorable histology these are noticeably different.

```
> summary(svyglm(rel~factor(stage)*factor(histol),design=dccs2,
                family=quasibinomial(log)))
```

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-2.7600	0.1167	-23.644	< 2e-16	***
factor(stage)2	0.7020	0.1790	3.922	9.30e-05	***
factor(stage)3	0.6730	0.1849	3.640	0.000285	***
factor(stage)4	1.0072	0.2207	4.564	5.58e-06	***
factor(histol)2	1.0344	0.2690	3.845	0.000127	***
factor(stage)2:factor(histol)2	0.1153	0.3405	0.339	0.734920	
factor(stage)3:factor(histol)2	0.4695	0.3118	1.505	0.132495	
factor(stage)4:factor(histol)2	0.4872	0.3316	1.469	0.142020	

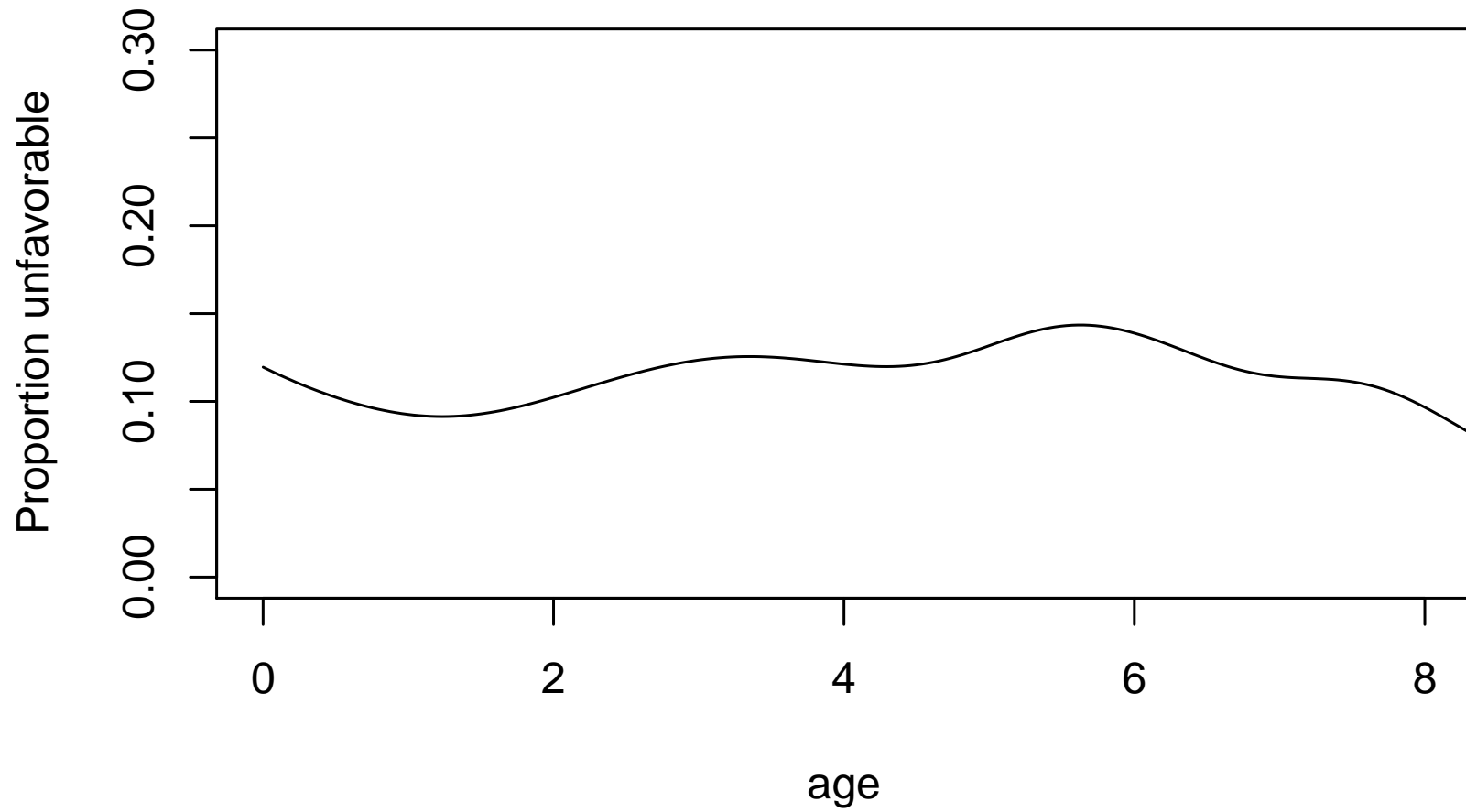
Computation in R

Other exploratory and descriptive analyses are also possible: how does histology vary with age at diagnosis?

[Really needs more detailed classification to be useful analysis]

```
s<-svysmooth(I(histol==2)~I(age/12),design=dccs2,bandwidth=1)
plot(s,ylim=c(0,0.3),xlim=c(0,8),
      xlab="age",ylab="Proportion unfavorable")
```

Computation in R



Two-phase calibration

Two-phase samples with many covariates at phase 1 provide good opportunities for calibration of phase 2 to phase 1: wide range of auxiliary variables.

Calibration improves the efficiency of estimation for population totals, choosing auxiliary variables for other targets of inference requires writing them as sums.

Almost all interesting estimators are 'asymptotically linear', in large samples they can be written as the sum of their influence functions.

Good auxiliary variables need to be correlated with the influence function of the target statistic.

Computing

`calibrate()` also works on two-phase design objects

Since the phase-one data are already stored in the object, there is no need to specify population totals when calibrating.

It is necessary to specify `phase=2`.

Earlier we had a two-phase case-control design

```
dccs2<-twophase(id=list(~seqno,~seqno),
  strata=list(NULL,~interaction(rel,instit)),
  data=nwtco, subset=~incc2)
```

Calibrating it to 16 strata of relapse×stage×institutional histology:

```
gccs8<-calibrate(dccs2, phase=2,
  formula=~interaction(rel,stage,instit))
```


Logistic regression

As all the phase-one data are available we can also estimate sampling weights by logistic regression, as suggested by Robins, Rotnitzky & Zhao (JASA, 1994).

Either use `calibrate` with `calfun="rrz"` or `estWeights`.

`estWeights` takes a data frame with missing values as input and produces a corresponding two-phase design with weights estimated by logistic regression.

Choice of auxiliaries

The other heuristic gain from the calibration viewpoint is in choosing predictors for estimating π .

The regression formulation shows that the predictors should have strong linear relationships with $U_i(\theta)$.

If the estimating function $U_i(\theta)$ is of a form such as

$$z_i w_i (y_i - \mu_i(\theta))$$

then z_i is approximately uncorrelated with U_i

So, don't use a variable correlated with a phase-2 predictor as a calibration variable, use a variable correlated with the phase-2 influence function.

`estWeights()` can take a phase-one model as an argument and use the influence functions from that model as calibration variables.

Example

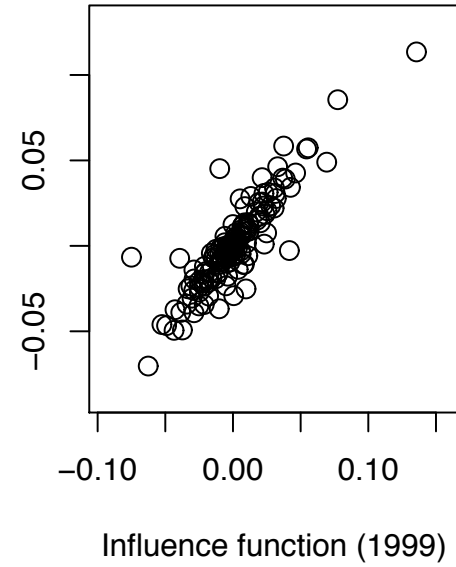
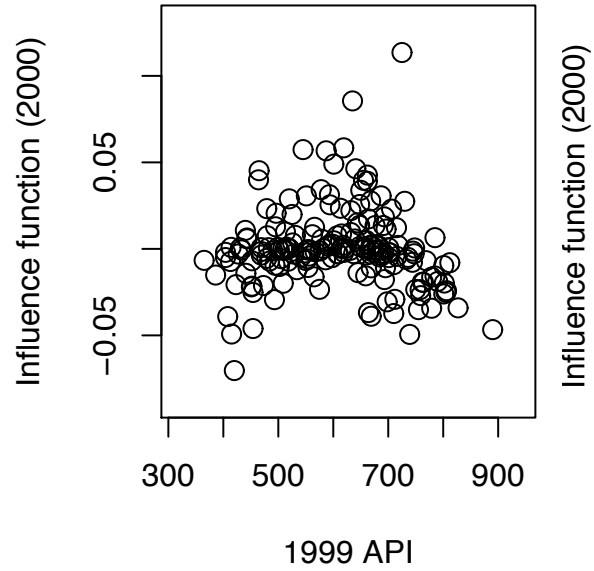
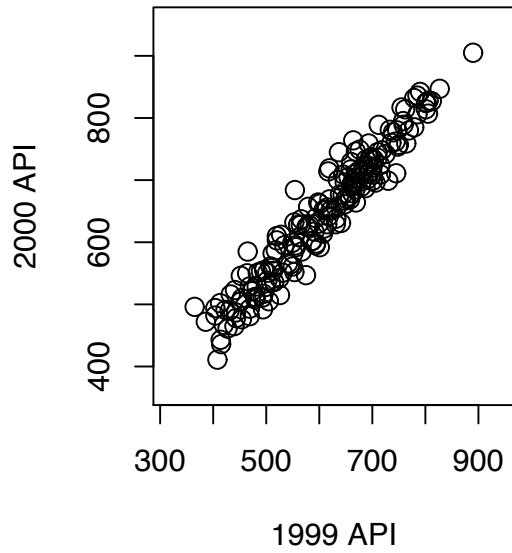
Single-phase sampling of California schools.

We want to fit a model with 2000 API as the outcome, with 2000 API measured only on a sample of schools.

We have population data on 1999 API and on the predictor variables. Calibrating using the raw variables has almost no effect.

Impute 2000 API by 1999 API and fit the model to complete imputed data. The parameter estimates in the imputed model are unreliable, but the influence functions still are good calibration variables.

Example: correlations



Example: code

```
> m0 <- svyglm(api00~ell+mobility+emer, clus1)
> var_cal <- calibrate(clus1, formula=~api99+ell+mobility+emer,
  pop=c(6194,3914069, 141685, 106054, 70366),
  bounds=c(0.1,10))
> m1<-svyglm(api00~ell+mobility+emer, design=var_cal)
>
> popmodel <- glm(api99~ell+mobility+emer, data=apipop,
  na.action=na.exclude)
> inffun <- dfbeta(popmodel)
> index <- match(apiclus1$snun, apipop$snun)
> clus1if <- update(clus1, ifint = inffun[index,1],
  ifell=inffun[index,2], ifmobility=inffun[index,3],
  ifemer=inffun[index,4])
> if_cal <- calibrate(clus1if,
  formula=~ifint+ifell+ifmobility+ifemer,
  pop=c(6194,0,0,0,0))
```

Example: code

```
> m2<-svyglm(api00~ell+mobility+emer, design=if_cal)
>
> coef(summary(m0))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	780.459500	30.0210123	25.997108	3.156974e-11
ell	-3.297892	0.4689026	-7.033215	2.173478e-05
mobility	-1.445370	0.7342887	-1.968395	7.473627e-02
emer	-1.814215	0.4233504	-4.285374	1.287085e-03

```
> coef(summary(m1))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	785.408240	13.7640081	57.062466	5.912274e-15
ell	-3.273108	0.6242978	-5.242864	2.756024e-04
mobility	-1.464732	0.6651257	-2.202188	4.989506e-02
emer	-1.676541	0.3742041	-4.480284	9.309647e-04

Example: code

```
> coef(summary(m2))
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	790.631553	5.8409844	135.359298	4.480786e-19
ell	-3.260976	0.1300765	-25.069679	4.678967e-11
mobility	-1.405554	0.2247022	-6.255187	6.214930e-05
emer	-2.240431	0.2150534	-10.418024	4.902863e-07

Two-phase calibration

Strategy

- Build a predictive model for the phase-two variables to impute at phase one
- Fit a phase-one model for the parameter of interested, using the imputed data
- Use the influence functions from the phase-one model to calibrate
- Fit the model to the phase-two sample

Wilms' Tumor

The imputation model is based on Kulich & Lin (2004).

```
impmodel <- glm(histol~instit+I(age>10)+I(stage==4)*study,
  data=nwts, subset=in.subsample, family=binomial)
nwts$imphist <- predict(impmodel, newdata=nwts, type="response")
nwts$imphist[nwts$in.subsample] <- nwts$histol[nwts$in.subsample]

ifmodel <- coxph(Surv(trel,relaps)~imphist*age+I(stage>2)*tumdiam,
  data=nwts)
inffun <- resid(ifmodel, "dfbeta")
colnames(inffun) <- paste("if",1:6,sep="")

nwts_if <- cbind(nwts, inffun)
if_design <- twophase(id = list(~1, ~1), subset = ~in.subsample,
  strata = list(NULL, ~interaction(instit, relaps)),
  data = nwts_if)
```

Wilms' Tumor

```
if_cal <- calibrate(if_design, phase=2, calfun="raking"  
  ~if1+if2+if3+if4+if5+if6+relaps*instit)
```

```
m1 <- svycoxph(Surv(trel, relaps)~histol*age+I(stage>2)*tumdiam,  
  design=nwts_design)
```

```
m2 <- svycoxph(Surv(trel, relaps)~histol*age+I(stage>2)*tumdiam,  
  design=if_cal)
```

```
m3 <- coxph(Surv(trel, relaps)~imphist*age+I(stage>2)*tumdiam,  
  data=nwts)
```

```
m4 <- coxph(Surv(trel, relaps)~histol*age+I(stage>2)*tumdiam,  
  data=nwts)
```

Wilms' Tumor

	Two-phase sample			full data
	sampling weights	raked	direct imputation	
Coefficient estimate				
histology	1.808	2.113	2.108	1.932
age	0.055	0.101	0.101	0.096
stage > 2	1.411	1.435	1.432	1.389
tumor diameter	0.043	0.061	0.061	0.058
histology:age	-0.116	-0.159	-0.159	-0.144
stage> 2:diameter	-0.074	-0.084	-0.083	-0.079
Standard error				
histology	0.221	0.171	0.174	0.157
age	0.023	0.014	0.016	0.016
stage > 2	0.361	0.276	0.249	0.250
tumor diameter	0.021	0.016	0.014	0.014
histology:age	0.054	0.039	0.040	0.035
stage > 2:diameter	0.030	0.022	0.020	0.020

PPS sampling

Options for PPS sampling

- A proposal by Brewer that turns out to be exactly what happens if you feed PPS data into the recursive variance algorithm for multistage data. Works for multistage sampling with PPS at any of the stages.
- Full unbiased variance estimator: you supply the matrix π_{ij}
- Hartley–Rao approximation: you supply $\sum_{i=1}^N \pi_i^2/n$, or R will estimate it by $\sum_{i=1}^n \pi_i/n$
- Overton's approximation (used in spatial sampling)

All but the first have Horvitz–Thompson and Yates–Grundy versions.

Computation

Brewer's approximation just comes for free from the recursive variance estimator.

The others methods compute the (approximate) covariance matrix of sampling indicators Δ and then compute either

$$\widehat{\text{var}}_{HT} = \sum_{i,j=1}^n \check{\Delta}_{ij} \check{x}_i \check{x}_j$$

or

$$\widehat{\text{var}}_{YG} = \sum_{i,j=1}^n \check{\Delta}_{ij} (\check{x}_i - \check{x}_j)^2$$

Where possible, $\check{\Delta}$ is stored as a sparse matrix (eg stratified sampling).

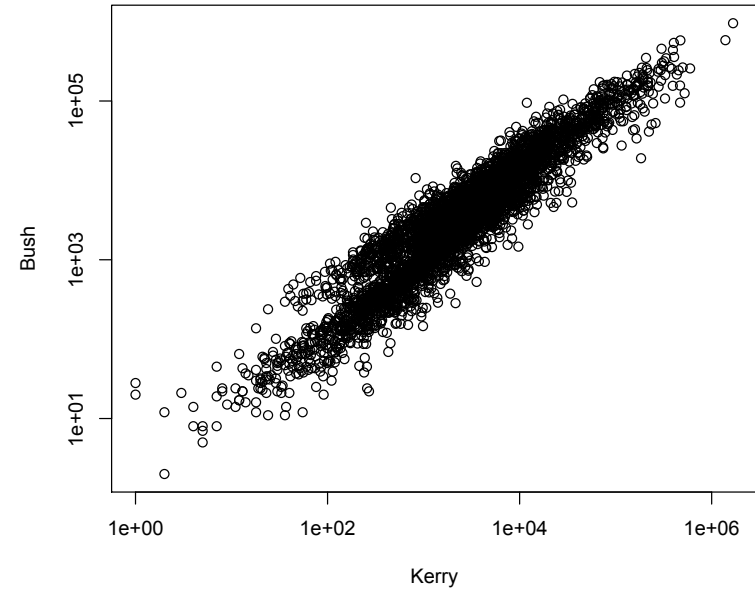
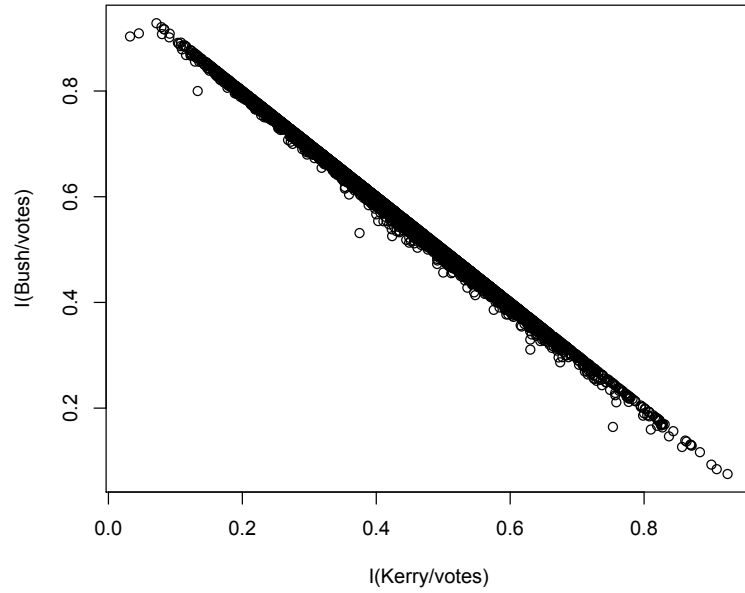
Example: sample of election data

This shows the code, it also illustrates why PPS sampling is useful. Although proportions of votes for Bush and Kerry are -vely correlated, totals are strongly correlated because large counties have more voters

```
data(election)
## high positive correlation between totals
plot(Bush~Kerry,data=election,log="xy")
## high negative correlation between proportions
plot(I(Bush/votes)~I(Kerry/votes), data=election)
```

We are using sample of just 40 counties, sampled proportional to total number of votes.

Example: sample of election data



Example: sample of election data

Declaring PPS sampling typically requires extra information

Yves Tillé's sampling package will compute joint sampling probabilities for many sampling methods: use the `ppsmat()` function to wrap the pairwise sampling probabilities for the sample.

The Hartley–Rao approximation requires the sum of squares of the marginal sampling probabilities, over the population, not just the sample. Wrap these with the `HR()` function

Since we are doing sampling without replacement, the marginal sampling probabilities must also be supplied to the `fpc` argument.

Example: sample of election data

```
## Horvitz-Thompson type
dpps_ht<- svydesign(id=~1, fpc=~p, data=election_pps,
  pps=ppsmat(election_jointprob))
dpps_hr<- svydesign(id=~1, fpc=~p, data=election_pps,
  pps=HR(sum(election$p^2)/40))
## Yates-Grundy type
dpps_yg<- svydesign(id=~1, fpc=~p, data=election_pps,
  pps=ppsmat(election_jointprob),variance="YG")
dpps_hryg<- svydesign(id=~1, fpc=~p, data=election_pps,
  pps=HR(sum(election$p^2)/40),variance="YG")
```

Example: sample of election data

Brewer's approximation and the Overton approximation don't require any extra information, just the marginal probabilities for the sample, supplied as `fpc`

```
## Horvitz-Thompson type
dpps_br<- svydesign(id=~1, fpc=~p, data=election_pps, pps="brewer")
dpps_ov<- svydesign(id=~1, fpc=~p, data=election_pps, pps="overton")
dpps_hr1<- svydesign(id=~1, fpc=~p, data=election_pps, pps=HR())
```

And we can always use a with-replacement approximation by omitting the `fpc` argument

```
dppswr <-svydesign(id=~1, probs=~p, data=election_pps)
```

Example: sample of election data

The true population totals are

Bush	Kerry	Nader
59645156	56149771	404178

```
> svytotal(~Bush+Kerry+Nader, dpps_ht)
```

	total	SE
Bush	64518472	2604404
Kerry	51202102	2523712
Nader	478530	102326

```
> svytotal(~Bush+Kerry+Nader, dpps_yg)
```

	total	SE
Bush	64518472	2406526
Kerry	51202102	2408091
Nader	478530	101664

Example: sample of election data

```
> svytotal(~Bush+Kerry+Nader, dpps_hr)
```

	total	SE
Bush	64518472	2624662
Kerry	51202102	2525222
Nader	478530	102793

```
> svytotal(~Bush+Kerry+Nader, dpps_hryg)
```

	total	SE
Bush	64518472	2436738
Kerry	51202102	2439845
Nader	478530	102016

```
> svytotal(~Bush+Kerry+Nader, dpps_br)
```

	total	SE
Bush	64518472	2447629
Kerry	51202102	2450787
Nader	478530	102420

Example: sample of election data

```
> svytotal(~Bush+Kerry+Nader, dpps_ov)
```

	total	SE
Bush	64518472	2939608
Kerry	51202102	1964632
Nader	478530	104373

```
> svytotal(~Bush+Kerry+Nader, dpps_hr1)
```

	total	SE
Bush	64518472	2472753
Kerry	51202102	2426842
Nader	478530	102595

```
> svytotal(~Bush+Kerry+Nader, dppswr)
```

	total	SE
Bush	64518472	2671455
Kerry	51202102	2679433
Nader	478530	105303

That's all, folks

Any questions?