The survey package

Thomas Lumley

UW Biostatistics
R Core Development Team

Vanderbilt — 2004-2-16/17

Overview: survey package for R

- Free software (LGPL) with extensible design
- Handles stratification, clustering, unequal-probability sampling, post-stratification
- Provides Taylor linearization and replicate-weight variances (jackknife, BRR, bootstrap user-supplied weights). Next version handles multistage finite-population corrections.
- Good coverage of regression, requires programming for complex tables.
- Relatively slow.
- New, and therefore less well tested.

Interactive data analysis

John Chambers ACM Software Systems award citation:

For the S system, which has forever altered how people analyze, visualize, and manipulate data.

Interactive analysis is not always appropriate

- Many non-statisticians do not have the expertise to benefit from interactive analysis
- Even expert statistical analysis should often be planned and programmed in advance, executed quickly when data become available.

R package system

- Quality control: R CMD check requires that every function is documented, the documented arguments match the actual ones, the examples all run, etc. Also supports extra validation/regression tests.

- Centralized distribution: Comprehensive R Archive Network (`http://cran.r-project.org`)

- Easy to install: `install.packages()`, menu items in GUI versions of R

- Cross-platform support: Under Unix, R knows what compiler and linker flags are needed. For Mac OS X and Windows, compiled binary versions of CRAN packages are created automatically.

Challenges in integrating survey software

- Probability samples come with a lot of meta-data that has to be linked correctly to the observations

- Interesting data sets tend to be large.

- Design-based inference literature is largely separate from rest of statistics, doesn't seem to have a unifying concept such as likelihood to rationalize arbitrary choices.

In part, too, the specialized terminology has formed barriers around the territory. At one recent conference, I heard a speaker describe methods used in his survey with the sentence, "We used BRR on a PPS sample of PSU's, with MI after NRFU"

(Sharon Lohr, American Statistician 5/2004)

Progress

- Thanks to Moore's Law, desktop computers can now handle large data sets

- High-level languages simplify the meta-data problem

- Model-based inference is starting to use similar mathematical techniques to design-based inference: inverse-probability weighting, sandwich variance estimates, subsampling bootstrap, . . .

- Semiparametric inference for statistical functionals is philosophically more similar to design-based inference: parameters are defined as the result of a computation on a (theoretical or empirical) distribution function.

Analysis Methods

- Estimation by inverse-probability weighting:
  - summary statistics, totals, ratios
  - linear regression, GLMs, Cox model
  - contingency tables.
- Variances by linearisation.
- Variances by replicate weights

Statistical functionals

Define the average slope of Y with X as

$$\beta = \sum_{i \neq j} w_{ij} \frac{y_i - y_j}{x_i - x_j}$$

which is the least squares estimate if $w_i \propto (x_i - x_j)^2$ over any finite sample or population, and infinite populations can be handled by changing the sum to an integral. There is no mathematical commitment to any model for (X, Y).

As for most parameters, β satisifies

$$\sum_i U_i(\beta) = 0$$

for some estimating function $U_i = U_i(x_i, y_i; \beta)$.

Probability weights

If the probability of obtaining observation i is π_i, and U_i is the estimating function for the population then

$$\sum_i \frac{1}{\pi_i} U_i(\beta) = \sum_i \tilde{U}_i(\beta) = 0$$

is an unbiased estimating equation for the sample.

Variances

If the population is large or the sample is with-replacement then $\sum \tilde{U}_i$ will have a Normal distribution. To get interval estimates we can

- Estimate the variance of $\sum_i \tilde{U}_i(\beta)$, which is just a mean, and apply the delta method (first-order Taylor series linearisation)
$$\mathrm{var}[\hat{\beta}] = E\left[\frac{\partial \tilde{U}}{\partial \beta}\right]^{-1} \mathrm{var}[\tilde{U}_i] E\left[\frac{\partial \tilde{U}}{\partial \beta}\right]^{-1}$$

- Perturb $\tilde{U}_i$ by changing weights to get approximate realisations from the distribution of $\sum \tilde{U}_i(\beta)$, and solve for β from each one (bootstrap, jackknife, replicate weights, subsampling).

(in fact even linearisation can be seen as infinitesimal perturbation)

Specifying survey design

```
des <- svydesign(id=~PSU, strata=~strata,
    fpc=~Pop.size, weight=~weights,
    data=dataset)
```

Creates an object containing the survey design information. Features such as strata or finite population correction are optional.

Sampling weights can be specified as weights or probabilities, and for multi-stage sampling can also be specified as probabilities at each stage

Finite population correction can be specified as proportion or population size, per observation or per stratum (currently restricted to first stage of sampling).

Specifying replicate weights

```
rdes <- svrepdesign(dataset, weights=~weights,
    repweights=repweights, type="BRR")

adss<-svrepdesign(data = adssdata,
    repweights = adssdata[, 782:981],
    scale = 1, rscales = adssjack, type = "other",
    weights = ~PH1FW0, combined.weights = TRUE,
    fpc=adssfpc, fpctype="correction")

rdes<-as.svrepdesign(des)
```

Sampling weights can be combined with or separate from replicate weights.

If R does not know how the weights were constructed a per-observation (`rscales`) and overall (`scale`) multiplier are needed to convert sum of squared deviations into variance.

Specifying survey information

The survey design object contains the observed data and the meta-data describing the sampling. This object, rather than a data frame, is supplied to survey analysis functions.

Subsetting preserves the meta-data

```
first100.schools <- cal.schools[1:100,]
low.ses <- subset(cal.schools, ell>25 | meals >50)
```

and an `update` function allows adding variables

```
cal.schools <- update(cal.schools,
                      low.ses= ell>25 | meals >50,
                      APIchange = api00-api99)
```

Specifying survey information

Printing the object gives some summary information; more is available with the `summary` function:

```
> dstrat
Stratified Independent Sampling design
svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
    fpc = ~fpc)

> dclus2
2 - level Cluster Sampling design
With ( 40, 126 ) clusters.
svydesign(id = ~dnum + snum, weights = ~pw, data = apiclus2)

> dclus1
1 - level Cluster Sampling design
With ( 15 ) clusters.
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
```

Specifying survey information

```
> summary(subset(dstrat, stype != "H"))
Stratified Independent Sampling design
subset.survey.design(dstrat, stype != "H")
Probabilities:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.02262 0.02262 0.02262 0.03145 0.04912 0.04912
Stratum sizes:
             E  H  M
obs        100  0 50
design.PSU 100 50 50
actual.PSU 100  0 50
Population stratum sizes (PSUs):
  strata    N
1      E 4421
2      H  755
3      M 1018
Data variables:
 [1] "cds"      "stype"    "name"     "sname"    "snum"     "dname"    "dnum"
 [8] "cname"    "cnum"     "flag"     "pcttest"  "api00"    "api99"    "target"
...
```

Specifying survey information

Post-stratification and raking adjust sample distributions of a set of categorical variables to the known population margins (same issue as estimated vs prespecified weights in Robins-type causal inference models)

```
cal.ps<-postStratify(cal.schools,
   strata=~stype, population=pop.table)
```

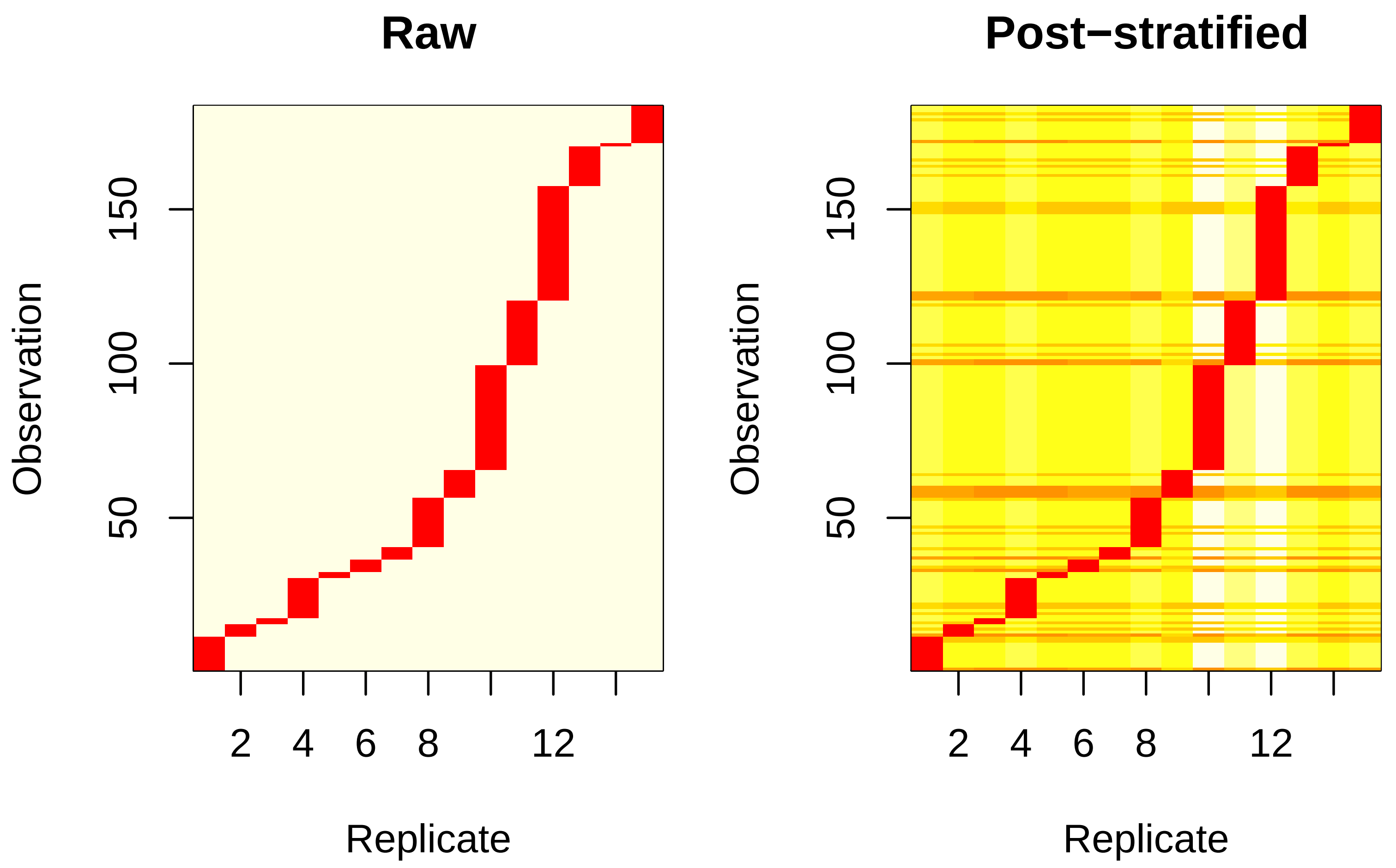
The `rake()` function is similar but takes a list of formulas and a list of population tables.

Specifying survey information

See the effect on weights with

```
image(cal.schools, type.="total")
image(cal.ps, type.="total")
```

Specifying survey information



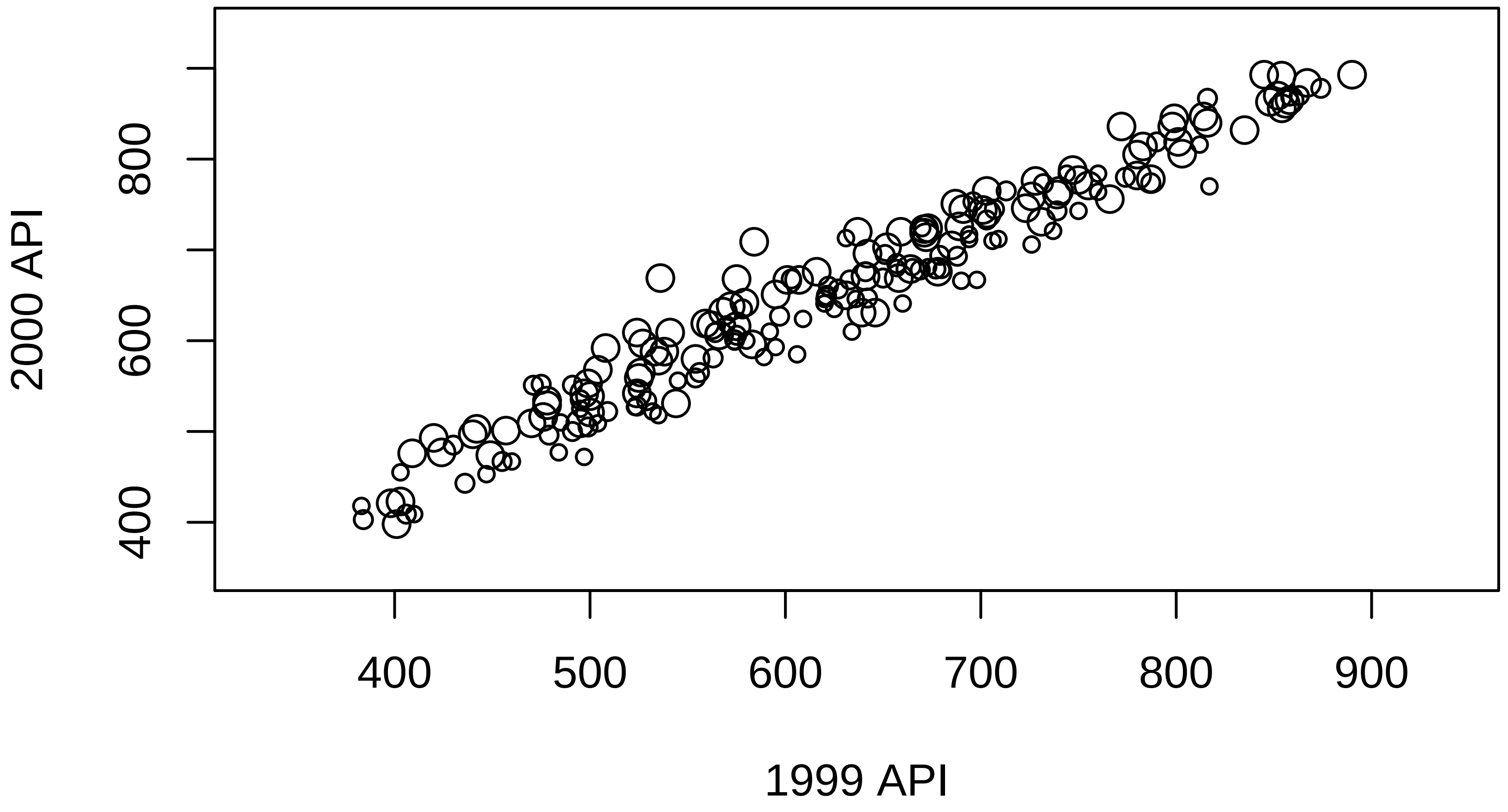
Graphics

Scatterplots allowing for sampling weights

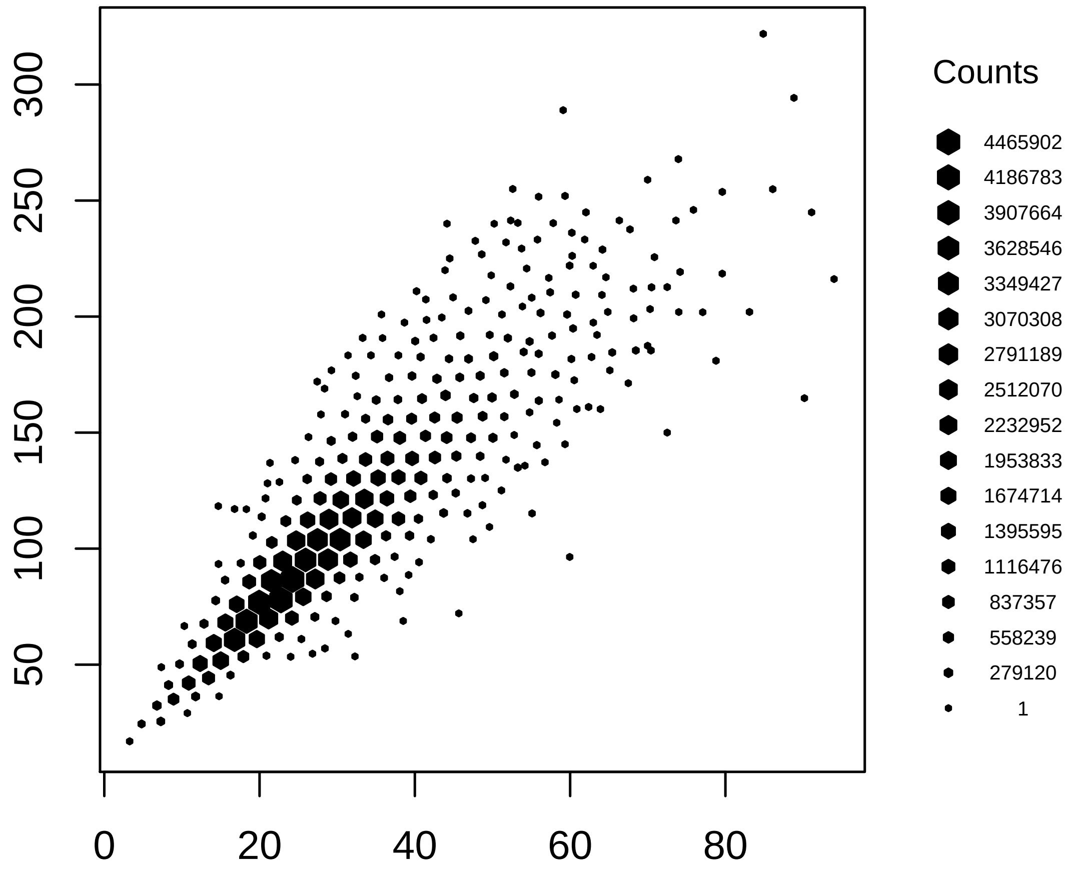
- Bubble plots (for small surveys)
- Hexagonal binning (for large surveys)
- Random subsampling (for large surveys)

Random subsampling can be used for other plot types, eg boxplots.

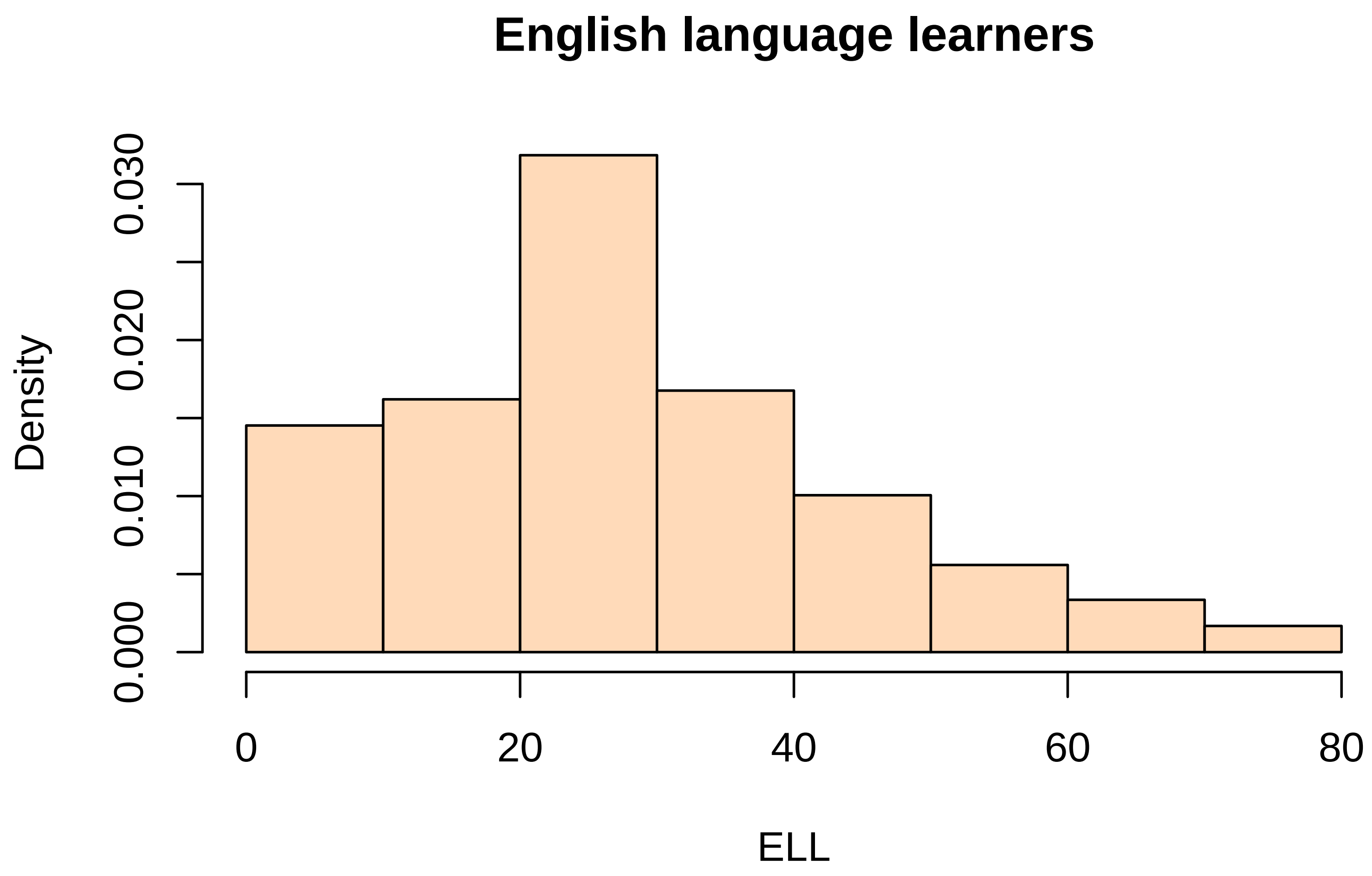
Graphics



Graphics


300
250
200
150
100
50
0
20
40
60
80
Counts
4465902
4186783
3907664
3628546
3349427
3070308
2791189
2512070
2232952
1953833
1674714
1395595
1116476
837357
558239
279120
1

Graphics


English language learners
Density
0.000
0.010
0.020
0.030
0
20
40
60
80
ELL

Design principles: interface

S has an established modelling interface where variables to be used in a computation are specified by a formula and a data frame.

The survey package uses this interface, but replaces the data frame with a survey object.

```
svymean(~api00, design=cal.schools)
svyratio(~api.stu, ~enroll, design=cal.schools)
svyglm(api00~meals+ell+mobility, design=cal.schools)
svytable(~sch.wide+stype, design=cal.schools)
svychisq(~sch.wide+stype, design=cal.schools)
svyby(~api00, by=~sch.wide+stype,
              design=cal.schools, svymean)
```

Design principles: interface

For regression models, the result of a model-fitting function is an object with methods to extract coefficients, variances, loglikelihood, AIC, residuals etc, as appropriate.

For summary statistics the result can be fed to `ftable` for prettier output: eg table of %ages for school type by success in meeting comparative improvement target.

```
> a <- svymean(~interaction(stype, comp.imp), design = dclus1)
> b <- ftable(a, rownames = list(stype = c("E", "H",
    "M"), comp.imp = c("No", "Yes")))
> round(100 * b, 1)
              stype    E    H    M
comp.imp
No        mean      17.5  3.8  6.0
          SE         2.6  1.6  2.5
Yes       mean      61.2  3.8  7.7
          SE         4.2  1.6  2.2
```

Design principles: interface

These functions return objects containing the results and other information that enables them to be printed attractively.

```
 > svyratio(~api.stu,~enroll,dclus1)
Ratio estimator: svyratio(~api.stu, ~enroll, dclus1)
Ratios=
           enroll
api.stu 0.8497087
SEs=
             enroll
api.stu 0.008386297
```

Here's the internal structure of the object

```
> str(svyratio(~api.stu,~enroll,dclus1))
List of 3
 $ ratio: num [1, 1] 0.85
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr "api.stu"
  .. ..$ : chr "enroll"
 $ var  : num [1, 1] 7.03e-05
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr "api.stu"
  .. ..$ : chr "enroll"
 $ call : language svyratio(~api.stu, ~enroll, dclus1)
 - attr(*, "class")= chr "svyratio"
```

Design principles: performance

Good design in a high-level interpreted language emphasizes simplicity of code rather than efficiency. Simple code is easier to get right. The survey package contains only about 3000 lines of code.

On my desktop the replicate-weight computations for a fairly large dataset are ten times faster than `NASSVAR` on a 1984 IBM mainframe. On my laptop they are only three times faster (the main difference is memory size).

Optimisation for speed and memory is a useful next step. The R profiler allows program bottlenecks to be identified in real use, so that optimisation can be directed sensibly. If necessary, parts of the program can be replaced by C or Fortran code.

Design principles: extensibility

The modelling functions (eg `svyglm`, `svrepglm`) call the ordinary modelling functions (eg `glm`) to obtain point estimates.

Standard error computations are localized in two functions (`svyCprod` and `svrVar`).

For Taylor linearization, `svyCprod` computes the variance of the estimating functions. The other component of the variance, the inverse of the expected derivative of the estimating functions, is already available as the model-based variance estimate.

Creating a new estimator involves working out the correct calls to the model-based version and to `svyCprod` or `svrVar`, or writing a wrapper for `withReplicates` or `svymle`.

Design principles: polymorphism

A valuable feature of S is that the same function can do different things with different objects (eg `print`, `summary`, `image`). The survey package does not take full advantage of this

- The initial design didn't consider replication weights, so we have parallel `svy` and `svrep` functions for everything. This is fixed in the next version, so eg `svymean` works on both types of survey object.

- In the traditional S method system dispatch can be based on only one method, and functions such as `plot` and `mean` are already doing this. They can't be made to also consider whether they have a survey design object as another argument. The newer S4 method system would allow this, but a lot of things would need to be rewritten.

Replication and arbitrary statistics

The `withReplicates` function allows any statistic to be programmed.

Two forms: a function of weights and data, or an expression: eg ratio estimate

```
withReplicates(scdrep,
       quote(sum(.weights*alive)/sum(.weights*arrests)))

withReplicates(scdrep,
       function(w,data) sum(w*data$alive)/sum(w*data$arrests))
```

(in this case could use `svyratio` instead)

Validation

- Comparisons to Stata for Taylor linearization methods
- Datasets from Levy & Lemeshow
- A couple of WesVar analyses.
- VPLX documentation examples
- The PEAS project at Napier (Scotland)

To add

Major:

- Score tests for regression models

Minor:

- Better interfaces for ratio/regression estimation of totals
- Survival curve estimation
- Perhaps score-based confidence intervals (eg, Binder 1991)
- Performance improvements (speed, memory)

(and probably things I haven't thought of)

Main references

- Levy & Lemeshow Sampling of Populations
- Korn & Graubard Analysis of Health Surveys
- Wolter Introduction to Variance Estimation
- Shao & Tu The Jackknife and Bootstrap
- David Binder's estimating function papers
- Rao & Scott's contingency table papers
- Valliant (JASA 1983) on post-stratification

Main references

- The Stata survey manual (which credits John Eltinge, Bureau of Labor Statistics for helping in the design)
- A Westat white paper by Brick, Morganstein, & Valliant on replication weights.