Tools for *Thinking* in Statistical Computation and Graphics: A 40-year journey from APL to SAS to R

Michael Friendly

QM Brownbag Seminar

November, 2010
RAND Corp. predicted what a home computer would look like in 2004.

“With a teletype interface and the Fortran language, the computer will be easy to use.”

Me: Dad, can we get one?

Dad:

What’s the steering wheel for?

Why is IKE in that picture?

Who speaks Fortran?
Prelude: First summer job (1962)

- Test department, Harcourt, Brace & World
- Job: Calculate 45 correlations among 10 tests, for n=500
- Tool: Monroe calculator
- Insight: There has to be a better way
  - $n, \sum x, \sum y, \sum x^2, \sum y^2, \sum xy$ can be calculated on a single pass
- Gripes:
  - Don’t they have an IT dept?
  - I could write this in Fortran!

Method:

| enter $X, Y$ | 10  | 5  |
| square      | 100 | 25 |
| sum         | $\sum x, \sum x^2, \sum xy, \sum y, \sum y^2$ |
• **CS view**: All programming languages can be proved to be equivalent (to a Turing machine)

• **Cognitive view**: Languages differ in
  - **expressive power**: ease of translating what you want to do into the results you want
  - **elegance**: how well does the code provide a human-readable description of what is done?
  - **extensibility**: ease of generalizing a method to wider scope
  - **learn-ability**: your learning curve (rate, asymptote)
## Programming languages: Power & elegance

<table>
<thead>
<tr>
<th>Language</th>
<th>Features:Tools for thinking?</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN</td>
<td>Subroutines – reusable code</td>
</tr>
<tr>
<td></td>
<td>Subroutine libraries (e.g., BLAS)</td>
</tr>
<tr>
<td>APL, APL2STAT</td>
<td>N-way arrays, nested arrays</td>
</tr>
<tr>
<td></td>
<td>Generalized reduction, outer product</td>
</tr>
<tr>
<td></td>
<td>Function operators</td>
</tr>
<tr>
<td>Logo</td>
<td>Turtle graphics</td>
</tr>
<tr>
<td></td>
<td>Recursion, list processing</td>
</tr>
<tr>
<td>Lisp, LispStat, ViSta</td>
<td>Object-oriented computing</td>
</tr>
<tr>
<td></td>
<td>Functional programming</td>
</tr>
<tr>
<td>Perl</td>
<td>Regular expressions</td>
</tr>
<tr>
<td></td>
<td>Search, match, transform, apply</td>
</tr>
<tr>
<td>SAS</td>
<td>??</td>
</tr>
<tr>
<td>R</td>
<td>??</td>
</tr>
</tbody>
</table>
Programming languages: Elegance - Logo

- Features:
  - Based on Lisp, but tuned to young minds
  - Turtle graphics: draw by directing a turtle, not by (x,y) coordinates
    - Analytic geometry rests on a coordinate system.
    - Turtle geometry is "body syntonic": Tell turtle what to do.
  - Data types: words, lists, arrays, property lists
  - Lists & list processing: inherited from Lisp, but with gentler syntax. Lists are infinitely expandable & nestable.
  - Recursion rather than iteration is the natural method to process lists
  - Extensions:
    - multiple, animated turtles (sprites);
    - object-oriented programming (message passing) -> SmallTalk
Logo : Turtle graphics

Turtle primitives: forward, back, left, right, penup, pendown, ...

Recursive procedures:

```
to spiral :size :angle
  if :size > 100 [stop]
  forward :size
  right :angle
  spiral (:size + 2) :angle
  end
end
```

Logo procedures: teach the turtle a new word

```
> to square :side
  repeat 4 [fd side rt 90]
  end

> square 100
```

```
spiral 0 90
spiral 0 91
```
Logo : Hilbert curves

Start with some basic shape

What happens if you replace each line with a smaller copy of the basic shape?

What happens if you continue this process?

Logo was more than just pretty pictures

to Hilbert0 :turn :size
  right :turn
  forward :size
  left :turn
  forward :size
  left :turn
  forward :size
  right :turn
end
Hilbert curve: A continuous, space-filling fractal, of Hausdorff dimension 2

Theorem (Hilbert, 1891): The euclidean length of the n-th depth Hilbert curve, $H_n$, is $2^n - \frac{1}{2^n}$

Proof (by enumeration): Redefine forward to calculate total turtle path length

```logo
to Hilbert :depth :turn :size
if :depth = 0 [stop]
right :turn
  Hilbert (:depth-1) :-turn :size
forward :size
left :turn
  Hilbert (:depth-1) :turn :size
forward :size
  Hilbert (:depth-1) :turn :size
left :turn
forward :size
  Hilbert (:depth-1) :-turn :size
right :turn
end
```

```logo
make "total:length :total:length + :size
forward :size
end
```
Programming languages: Power - APL

Quotes:

- APL2 is arguably the most powerful language yet developed for expressing statistical computation. One’s ability to get work done, however, depends as much on the programming environment as on the primitives of the language.

  Friendly & Fox, JCGS, 1994

- APL is the most powerful notation for array processing ever invented. Because of its lack of influence on other languages, it will not be discussed further.


- APL as “write-only” language: Saying “powerful language” is just a friendlier way of saying “obfuscated syntax”.

  Jim Lehmer

(2+\#0=I\*\.\|I)/I\+\|50    A prime numbers from 1..50
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems

A.N. Whitehead

e.g., matrix algebra (Arthur Cayley, ~1850): \( \mathbf{A} \mathbf{x} = \mathbf{b} \rightarrow \mathbf{x} = \mathbf{A}^{-1} \mathbf{b} \)

Characteristics of computational notation (K. E. Iverson):

- Universality: Any problem in executable form \( \rightarrow \) unambiguous result
- Ease of expressing constructs arising in problems
- Suggestivity: expressions in one set of problems suggest others for application in other problems
- Subordination of detail, e.g., vectors \( \rightarrow \) n-way arrays, named functions
- Economy: Utility of a language as a tool for thought increases with the range of topics it can treat, but decreases with the size of vocabulary and complexity of grammatical rules the user must keep in mind
- Amenability to formal proofs: Extent to which notation facilitates proofs (by induction, exhaustion, etc.)
Many features, but three most important for statistical computation and graphics:

- n-way arrays, nested arrays
- implicit iteration with data and each (¨)
- operators: functions of functions
APL2, APL2STAT Features

- n-way arrays, nested arrays
  e.g. partitioned matrices

\[
Z = (y | X) \quad Z'Z = (y | X)'(y | X) = \begin{bmatrix} y'y & y'X \\ X'y & X'X \end{bmatrix}
\]
**APL2STAT**: Datasets as nested arrays

**APL2STAT objects:**
- N x 2 nested arrays of (property, value)
- Value can be any APL2STAT object
- Inheritance of methods through TYPE, PARENT properties

---

Sample data set. The variables are:
1. Education (years)
2. Income ($000)
3. Gender ('M'/ 'F')
**APL2, APL2STAT Features**

- **implicit iteration with data and each**
  - APL has no built-in control structures (do-while, foreach, loops)
  - instead, powerful data-driven computations replace iteration
  - e.g., calculate cumulative proportions by rows or columns

```
÷×M+3 4ρ↓12
 1 2 3 4
 5 6 7 8
 9 10 11 12

+/[2]M                                       #sum over columns
 10 26 42

 0.1 0.2 0.3 0.4
 0.1923 0.2308 0.2692 0.3077
 0.2143 0.2381 0.2619 0.2857

+/M×[1]+/[2]M                                #accumulative proportions
 0.1 0.3 0.6 1
 0.1923 0.4231 0.6923 1
 0.2143 0.4524 0.7143 1
```
APL2, APL2STAT Features

- operators: functions of functions ("closures")
  - familiar example: derivatives
    \[
    f(x) = 3x^2 - 2x + 3
    \]
    \[
    f'(x) = \frac{d}{dx}f(x) = 6x - 2
    \]
    \[
    f''(x) = \frac{d}{dx}f'(x) = 6
    \]
  - APL primitive operators: f/ (reduction), f\ (scan) °.f (outer product)

---

```apl
# GENERALIZED OUTER PRODUCTS
1 2 3.*.+10 20 # a sum of each with each
11 21
12 22
13 23

1 2 3.*.1 2 3 # 13 raised to each power
1 1 1
2 4 8
3 9 27

1 2 3.*.2 1 2 3 # a lower triangular matrix
1 0 0
1 1 0
1 1 1
```
• APL2STAT operators: e.g., BOOTSTRAP
• APL2STAT operators: e.g., BOOTSTRAP

```
\(\text{Define a function to return mean and variance}\)
\[
[0] \ R+\text{STATS X} \\
[1] \ R+(\text{MEAN X}),\text{VARIANCE X}
\]

\(\text{DATA}++\ \text{NORMAL\_RAND 50} \ \text{\# 50 lognormal values}\)

\(\text{R+STATS BOOTSTRAP DATA} \ \text{\# bootstrap means and variances}\)

\(\text{Result for full sample: 1.5357 3.5881}\)

\(\text{Beginning BOOTSTRAP replications}\)

\(\text{BOOTSTRAP replications completed.}\)

\(\text{('MEAN', 'VAR'), [1]5+[1]R} \ \text{\# first 5 bootstrap replications}\)

\(\text{MEAN} \ \text{VAR}\)
\(1.5824 \ 3.8957\)
\(1.4367 \ 2.6835\)
\(2.1826 \ 7.8176\)
\(1.9824 \ 5.5228\)
\(1.174 \ 1.1709\)

\(\text{DESCRIBE R} \ \text{\# summarize}\)

\(\text{Col_1} \ \text{Col_2}\)
\(\text{Mean} \ 1.5651 \ 3.7373\)
\(\text{Standard deviation} \ 0.25711 \ 1.3982\)
\(\text{Minimum} \ 1.126 \ 1.1541\)
\(\text{Lower hinge (Q1)} \ 1.381 \ 2.651\)
\(\text{Median} \ 1.5329 \ 3.652\)
\(\text{Upper hinge (Q3)} \ 1.7305 \ 4.6877\)
\(\text{Maximum} \ 2.3215 \ 7.8176\)
\(\text{N (selected, *'.'*)} \ 100 \ 100\)
Statistical computing: common tasks

• Data summaries by individual or group(s)
  ▪ Find mean, sd, Q25, Q75 for each group
  ▪ Group-wise transformations (scale, standardize)
  ▪ Fit same model to each S or group
    ▪ Longitudinal data: individual trajectories
    ▪ Multilevel models: level 1 models
  ▪ What’s the pattern?
    ▪ How does your software (SAS or R) help you think about solutions?
More common tasks

• Simulation studies
  ▪ Effect of violation of constant variance on \( p \)-value/power in one-way ANOVA
  ▪ Determining power for the new Cribbie-Mara multiple comparison procedure

• Experiment:
  ▪ Generate multiple datasets with varying parameters
  ▪ Analyze each: empirical \( p \)-value or power
  ▪ Summarize collection

• What’s the pattern?
  ▪ How does your software (SAS or R) help you think about solutions?
Some less common tasks

- Computer-intensive methods
  - Multiple-imputation for missing data
    - Generate $m$ imputed complete data sets
    - Analyze each using standard methods
    - Combine to give tests taking missing into account
  - Bootstrapping, when parametric methods fail
    - Generate $B$ bootstrap samples from the data
    - Obtain standard estimates for each
    - Combine to give bootstrap estimate & CI
  - What’s the pattern?
    - How does your software (SAS or R) help you think about solutions?
Some less common tasks

- Implement a new statistical procedure
  - Cribbie-Mara procedure; Flora’s $\Lambda$
  - Make them publicly available
- Implement a new graphical method
  - Fox: effect plots
  - mosaic displays
  - HE plots
- Write a paper using reproducible research methods
  - All data, results, graphs verifiable & public
- How does your software (SAS or R) help?
SAS thinking

• PROC steps, DATA steps, ODS & more

```sas
data class;
  input age sex ht wt;
datalines;
  20 M 75 152
  22 F 69 132

proc glm data=class;
  class sex;
  model wt = ht sex;
  output out=results
    p=predict r=resid;
results

proc gplot data=results;
  plot wt * ht = sex;
symbol1 ... symbol2 ...
```

But, wait: there’s more:
• ODS: capture *any* results to a data set
• ODS graphics: automatic plots from procs
• SAS/IML: matrix computations
• Macro language: custom graphics
• proc SQL, …
PROC SUMMARY

- General procedure for univariate summaries
- Output dataset for further processing

```
proc summary;
  var ...;
  class ...;
  output ...
results
```
### Cars data:

<table>
<thead>
<tr>
<th>Obs</th>
<th>make</th>
<th>model</th>
<th>mpg</th>
<th>cylinder</th>
<th>engine</th>
<th>horse</th>
<th>weight</th>
<th>accel</th>
<th>year</th>
<th>origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chev</td>
<td>chevelle</td>
<td>18</td>
<td>8</td>
<td>307</td>
<td>130</td>
<td>3504</td>
<td>12.0</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>buick</td>
<td>skylark</td>
<td>15</td>
<td>8</td>
<td>350</td>
<td>165</td>
<td>3693</td>
<td>11.5</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>plymouth</td>
<td>satellite</td>
<td>18</td>
<td>8</td>
<td>318</td>
<td>150</td>
<td>3436</td>
<td>11.0</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>amc</td>
<td>rebel</td>
<td>16</td>
<td>8</td>
<td>304</td>
<td>150</td>
<td>3433</td>
<td>12.0</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>ford</td>
<td>torino</td>
<td>17</td>
<td>8</td>
<td>302</td>
<td>140</td>
<td>3449</td>
<td>10.5</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>ford</td>
<td>galaxie</td>
<td>15</td>
<td>8</td>
<td>429</td>
<td>198</td>
<td>4341</td>
<td>10.0</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>chev</td>
<td>impala</td>
<td>14</td>
<td>8</td>
<td>454</td>
<td>220</td>
<td>4354</td>
<td>9.0</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>plymouth</td>
<td>fury</td>
<td>14</td>
<td>8</td>
<td>440</td>
<td>215</td>
<td>4312</td>
<td>8.5</td>
<td>70</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Univariate, multiple summary statistics

```plaintext
proc summary data=cars;
  var mpg;
  class origin;
  output out=results mean=mean stddev=stddev qrange=qrange q1=q1 q3=q3;
run;
```

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>origin</th>
<th><em>FREQ</em></th>
<th>mean</th>
<th>stddev</th>
<th>qrange</th>
<th>q1</th>
<th>q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>406</td>
<td>23.5146</td>
<td>7.81598</td>
<td>11.5</td>
<td>17.5</td>
<td>29.0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>254</td>
<td>20.0835</td>
<td>6.40289</td>
<td>9.0</td>
<td>15.0</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>73</td>
<td>27.8914</td>
<td>6.72393</td>
<td>6.7</td>
<td>24.0</td>
<td>30.7</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>79</td>
<td>30.4506</td>
<td>6.09005</td>
<td>8.7</td>
<td>25.4</td>
<td>34.1</td>
</tr>
</tbody>
</table>
Multivariate: one summary measure

```
proc summary data=cars;
  var mpg accel weight;
  class origin;
  output out=means mean=;
run;
```

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>origin</th>
<th><em>FREQ</em></th>
<th>mpg</th>
<th>accel</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>406</td>
<td>23.5146</td>
<td>15.5197</td>
<td>2979.41</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>254</td>
<td>20.0835</td>
<td>14.9425</td>
<td>3372.70</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>73</td>
<td>27.8914</td>
<td>16.8219</td>
<td>2431.49</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>79</td>
<td>30.4506</td>
<td>16.1722</td>
<td>2221.23</td>
</tr>
</tbody>
</table>

Univariate: two+ class variables;

```
proc summary data=cars;
  var mpg;
  class origin cylinder;
  output out=means mean=;
run;
```

<table>
<thead>
<tr>
<th><em>TYPE</em></th>
<th>origin</th>
<th>cylinder</th>
<th><em>FREQ</em></th>
<th>mpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>.</td>
<td>406</td>
<td>23.5146</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>3</td>
<td>4</td>
<td>20.5500</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>.</td>
<td>73</td>
<td>27.8914</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>.</td>
<td>79</td>
<td>30.4506</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>4</td>
<td>72</td>
<td>27.8403</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>6</td>
<td>74</td>
<td>19.6635</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>8</td>
<td>108</td>
<td>19.9857</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>4</td>
<td>66</td>
<td>28.4111</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>5</td>
<td>3</td>
<td>27.3667</td>
</tr>
<tr>
<td></td>
<td>E</td>
<td>6</td>
<td>4</td>
<td>20.1000</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>3</td>
<td>4</td>
<td>20.5500</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>4</td>
<td>69</td>
<td>31.5957</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>6</td>
<td>6</td>
<td>23.8833</td>
</tr>
</tbody>
</table>
• **BY processing**
  - All (almost) SAS procedures accept **BY** stmt
  - Do the procedure for all values of **BY** variables

```sql
proc mi data=foo;
var y x1-x3;
proc reg outest=parm;
model y = x1-x3;
by _Imputation_;
proc mianalyze;
```
SAS thinking: Macros

- Macro language
  - Combine any number of PROC and DATA steps into a general procedure

```sas
%meanplot(data=cars, var=mpg, class=cylinder origin);
```
SAS thinking: Macros

```
%macro meanplot (data=_last_,
var=, class=, out=, z=1, ...);

proc summary data=&data;
  var &var;
  class &class;
  output out=&out mean= stderr=se;
proc glm data=&data outstat=_stat_;
  class &class;
  model &var = &class;
  /* extract MSE and dfe */
/* annotate data set to draw err bars */
data _bars_; set &out;
  x=&xvar; y=mean+se; function='move';
x=&xvar; y=mean-se; function='draw';
...  proc gplot data=&out;
  by &panels;
  plot &var * &xvar = &sym /
    anno=_bars_ &haxis &vaxis ...;
...  /* clean up */
proc datasets;
delete _work_ _bars_ ...;
```
• What’s wrong with macros?
  ▪ Another language to learn
  ▪ Text substitution, not computation
  ▪ Difficult to use other macros as building blocks
    ▪ No require() feature
    ▪ No version control
  ▪ No standards for documentation, examples, dissemination
SAS thinking: many languages

SAS/Graph:
• procs, Annotate language

Base SAS, SAS/STAT
• data step, proc steps
SAS thinking: many languages

%macro language
proc iml
  • matrix language, graphics
SAS/Graph:
  • procs, Annotate language
Base SAS, SAS/STAT
  • data step, proc steps
SAS thinking: many languages

Output delivery system (ODS)
- template language

%macro language

proc iml
- matrix language, graphics

SAS/Graph:
- procs, Annotate language

Base SAS, SAS/STAT
- data step, proc steps
What features contribute to the power of R for statistical computing & graphics?

Language & data
- Data objects: arrays, lists, data frames, ...
- Object methods (S3, S4)
- Formulas: compact notation for models and graphs
- Grammars for graphics: base, lattice, ggplot2
- All of the above are extensible!

R environment
- Documentation: .Rd format, executable examples, vignettes
- Packages: now over 2000 contributed packages
- CRAN: easy upgrade, task views, ...
- Social: newsgroups (R-help), blogs, galleries, ...
• Data objects: data.frame, matrix, array, list, ...

• *Everything* in R is an object!

• Objects have *methods*

```r
> x <- 1:20
> y <- 10 + 3*x + 2*rnorm(20)
> mymod <- lm(y ~ x)

> class(mymod)
[1] "lm"
> methods(class = "lm")
[1] add1.lm* alias.lm* anova.lm case.names.lm*
[5] confint.lm* cooks.distance.lm* deviance.lm* dfbeta.lm*
[9] dfbetas.lm* drop1.lm* dummy.coef.lm* effects.lm*
[13] extractAIC.lm* family.lm* formula.lm* hatvalues.lm
[17] influence.lm* kappa.lm labels.lm* logLik.lm*
[21] model.frame.lm model.matrix.lm plot.lm predict.lm
[25] print.lm proj.lm* residuals.lm rstandard.lm
[29] rstudent.lm simulate.lm* summary.lm variable.names.lm*
[33] vcov.lm*
```

Non-visible functions are asterisked
```r
> coefficients(mymod)
(Intercept)           x
  9.831           3.039

> residuals(mymod)
   1    2    3    4    5    6    7    8    9   10
 1.6690 -1.1390 -0.4393 -3.6188  4.6582  2.0145 -1.1890  2.0841 -3.1287  2.1334
 11   12   13   14   15   16   17   18   19   20
-0.2631 -2.1389 -1.2886 -2.2416  0.6391 -1.5430 -0.4538  2.4677  0.9650  0.8128

> op <- par(mfrow=c(1,4))
> plot(mymod)
> par(op)

plot(lm) objects: ‘regression quartet’
```
> library(car)
> class(Prestige)
[1] "data.frame"
> plot(Prestige[,1:4])
thinking: Methods

> library(vcd)
> class(HairEyeColor)
[1] "table"
> str(HairEyeColor)
  table [1:4, 1:4, 1:2] 32 53 10 3 11 50 10 ... - attr(*, "dimnames")=List of 3
  ..$ Hair: chr [1:4] "Black" "Brown" "Red" "Blond"
  ..$ Eye : chr [1:4] "Brown" "Blue" "Hazel" "Green"
  ..$ Sex : chr [1:2] "Male" "Female"
> plot(HairEyeColor, shade=TRUE)
thinking: Formulas

- Model formulas: response ~ predictor(s)
  - + adds new terms: \( y \sim x_1 + x_2 + x_3 \)
  - - omits terms: \( y \sim -1 + x \)
  - : interactions between terms: \( y \sim x_1 + x_2 + x_1:x_2 \)
  - * expands to interactions + terms (\( y \sim a*b \rightarrow y \sim a+b+a:b \))
  - \(^n\) all terms and interactions up to order \( n \):
    - \( y \sim (a+b+c)^2 \rightarrow y \sim a + b + c + a:b + a:c + b:c \)
    - \( y \sim (a+b+c)^3 \rightarrow y \sim a + b + c + a:b + a:c + b:c + a:b:c \)
  - functions: \( \log(x) \), \( I(x^2) \), \( \text{poly}(x, 4) \), ...
  - multivariate responses: \( \text{cbind}(y_1, y_2, y_3) \sim x_1 + x_2 + x_3 \)
  - short-hands:
    - Use everything else: \( y \sim . \)
    - Update methods: \( \text{update(model1, . ~ . + x5)} \)
• **generality**: applies to *all* model functions (with extensions)
  - Linear models: `lm()`
  - Generalized linear models: `glm()`
    - `glm(Freq ~ (row+col+layer)^2, family=poisson)`
  - Nonlinear models: `nls()`
    - `nls(y ~ Asym/(1 + exp((Xmid - log(conc))/Scale)))`
  - Generalized non-linear models: `gnm()`
    - `gnm(Freq ~ row+col + Diag(row,col) + Mult(row,col))`
  - Robust linear models: `MASS::rlm()`
  - Mixed models: `nmle`
thinking: Formulas

- **suggestivity**: a notation for other things
  - Crosstabs: `xtabs()`, `vcd::structable()`, ...

```r
> DF <- as.data.frame(UCBAdmissions) # make a data frame
> ## Nice for taking margins ...
> xtabs(Freq ~ Gender + Admit, DF)
   Admit
  Gender    Admitted Rejected
    Male       1198     1493
    Female     557     1278
> ## And for testing independence ...
> summary(xtabs(Freq ~ ., DF))
Call: xtabs(formula = Freq ~ ., data = DF)
Number of cases in table: 4526
Number of factors: 3
Test for independence of all factors:
  Chisq = 2000.3, df = 16, p-value = 0
```
• suggestivity: ...

- Formulas for graphs

```r
plot(logIMR ~ logGDP, data=UN)  # scatterplot
plot(logIMR ~ region, data=UN)  # boxplots
plot(logIMR ~ logGDP + contraception + educationFemale, data=UN) # 3 scatterplots
plot(~ logIMR + logGDP + contraception, data=UN)  # scatterplot matrix
```
Graph formulas: Lattice extensions

- Conditioned plots: $y \sim x \mid z$

```R
library(lattice)
data(Chem97, package = "mlmRev")bwplot(gcsescore ~ gender | factor(score), Chem97, layout = c(6, 1))
```
Graph formulas: Lattice extensions

- Conditional+grouping: \( y \sim x \mid z \), groups=

\[
data(BodyWeight, package = "nlme")
xyplot(weight ~ Time | Diet, BodyWeight, groups = Rat,
       type = "l", layout = c(3, 1))
\]
• Functional programming: apply a function to subsets of an object

\[
\text{apply}(X, \text{MARGIN}, \text{FUN}, \ldots)
\]

- matrix or array
- function applied
- subscript(s) applied over
- other args to FUN()

array of \(\text{dim}(X)[\text{MARGIN}]\)

• Other variations:
  - `lapply(X, FUN, \ldots)` – apply \(\text{FUN}(X[i], \ldots)\) to lists
  - `tapply(X, INDEX, FUN, \ldots)` – tables indexed by factors
  - convenience functions: `sweep()`, `by()`, `aggregate()`, `replicate()`, …
apply() examples

Create some data

```r
> # 100 random chisq(4) values in a 25x4 matrix
> dat <- matrix(rchisq(100, 4), ncol = 4)
> head(dat, 3)
[1,] 5.3343 3.2368 3.0089 7.7839
[3,] 2.8049 1.6793 8.9917 5.2745
```

apply some functions

```r
> apply(dat, 2, mean)
> # trimmed means
> apply(dat, 2, mean, trim=0.05)
> # variances
> apply(dat, 2, function(x) sd(x)^2)
[1]  5.1259  3.7888 10.2617  5.4181
```

Custom functions

```r
> skewness <-function (x) {
+     n <- length(x)
+     x <- x - mean(x)
+     skew <- sqrt(n) * sum(x^3)/(sum(x^2)^(3/2))  # std skewness
+     skew <- skew * sqrt(n * (n - 1))/(n - 2)     # SAS, SPSS form
+     skew
+ }
> apply(dat, 2, skewness)
[1] 0.87816 0.85000 0.72138 0.41406
```
simulate distribution of eigenvalues of a correlation matrix (Horn’s method)

```r
> set.seed(1)
> N <- 100
> dat <- mvrnorm(N, rep(0, 3), diag(3))
> cor(dat)
     [,1]       [,2]       [,3]
[1,]  1.0000000 -0.00099432  0.01838219
[2,] -0.00099432  1.00000000 -0.04953621
[3,]  0.01838219 -0.04953621  1.00000000
> eigen(cor(dat))$values
[1] 1.053170 0.999350 0.947479

> eigensim <- function(N, mu, sigma) {
+   dat <- mvrnorm(N, mu, sigma)
+   eigen(cor(dat))$values
+ }

> mu <- rep(0, 4)
> sigma <- diag(4)
> replicate(5, eigensim(N, mu, sigma))
[1,] 1.168631 1.223235 1.261902 1.228980 1.237303
[2,] 1.104111 1.117801 1.033671 1.064980 1.128785
[3,] 0.887435 0.916802 0.924390 0.960071 0.910789
[4,] 0.839831 0.742172 0.780051 0.745970 0.723140
```

Do it once, by hand

Wrap in a function

Replicate 5 times
replicate() examples

Do 1000 replications, summarize and plot

```r
> reps <- replicate(1000, eigensim(N, mu, sigma))
> (eigen <- apply(reps, 1, mean))
[1] 1.22589 1.05877 0.93298 0.78236
> (sdbars <- apply(reps, 1, sd))
[1] 0.073541 0.048007 0.046980 0.067464

> plot(eigen, type='b', pch=16, cex=1.5, ylim=c(0.5,1.5))
> abline(h=1, lty=2, col="green")
> segments(1:4, eigen+sdbars, 1:4, eigen-sdbars, col="blue")
```
• General package for split-apply-combine

**Split** a data object into pieces

**Apply** a function to each piece

**Combine** the pieces back together

---

**Input**
- a (array)
- d (data frame)
- l (list)

**Output**
- fun( ) -> 
- a (array)
- d (data frame)
- l (list)
- _ (nothing)

---

```
ply
```
Basic plyr functions

<table>
<thead>
<tr>
<th>output</th>
<th>array</th>
<th>data frame</th>
<th>list</th>
<th>discarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>aaply</td>
<td>adply</td>
<td>alply</td>
<td>a_ply</td>
</tr>
<tr>
<td>array</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data frame</td>
<td>daply</td>
<td>ddply</td>
<td>dlply</td>
<td>d__ply</td>
</tr>
<tr>
<td>list</td>
<td>laply</td>
<td>ldply</td>
<td>llply</td>
<td>l_ply</td>
</tr>
</tbody>
</table>

Arguments

a*ply(.data, .margins, .fun, ...)
d*ply(.data, .variables, .fun, ...)
l*ply(.data, .fun, ...)
• Advantages over std *apply functions
  ▪ consistent names, arguments and outputs
  ▪ convenient parallelization through the foreach package (large simulations: multi processors)
  ▪ input from and output to data.frames, matrices and lists
  ▪ progress bars to keep track of long running operations
  ▪ built-in error recovery, and informative error messages (failwith= argument)
  ▪ labels that are maintained across all transformations
new <- ddply(.data, .variables, .fun, ...)  

- **Arguments:**  
  - *data*: data frame to process  
  - *variables*: combinations of variables to split by  
  - *fun*: function to call on each piece  
  - ...: extra args passed to *fun()*  

- **Variable syntax:**  
  - Character: `c("sex", "year")`  
  - Numeric: `1:3`  
  - Formula: `~ sex + year`  
  - Special  
    - *(sex, year)*  
    - *(first = substr(name, 1, 1))*
plyr: ddply()

Ways to split a data frame

> ddply(DF, .(sex), "nrow")
  sex nrow
1 Female 3
2 Male 3

> ddply(DF, .(sex, age), "nrow")
  sex age nrow
1 Female 13 1
2 Female 14 1
3 Female 15 1
4 Male 13 2
5 Male 14 1

> ddply(DF, .(sex), summarize, mean.age=mean(age))
  sex mean.age
1 Female 14.00000
2 Male 13.33333

> ddply(DF, .(sex), summarize, mean.age=mean(age),
  sd.age=sd(age))
  sex mean.age  sd.age
1 Female 14.00000 1.0000000
2 Male 13.33333 0.5773503
plyr: a*ply()

Ways to split a 2-way array

- `.margin = c()`
- `.margin = 1`
- `.margin = 2`
- `.margin = 1:2`

Ways to split a 3-way array
plyr: aaply() examples

> dim(HairEyeColor)
[1] 4 4 2
> str(HairEyeColor)
attr(*, "dimnames")=List of 3
..$ Hair: chr [1:4] "Black" "Brown" "Red" "Blond"
..$ Eye : chr [1:4] "Brown" "Blue" "Hazel" "Green"
..$ Sex : chr [1:2] "Male" "Female"

> # one-way marginal frequencies
> aaply(HairEyeColor, 1, sum)
Black Blond Brown   Red
108   127   286    71
> aaply(HairEyeColor, 2, sum)
Blue Brown Green Hazel
215  220  64  93
> aaply(HairEyeColor, 3, sum)
Female   Male
313    279

> # collapse over Sex
> (HE <- aaply(HairEyeColor, 1:2, sum))
Eye
Hair  Blue Brown Green Hazel
Black    20    68    5   15
Blond    94     7   16    10
Brown    84   119   29   54
Red      17    26    14   14

> percents <- function(x) x/sum(x)
> aaply(HE, 1, percents)
Hair          Blue      Brown     Green      Hazel
Black 0.1851852 0.62962963 0.0462963 0.13888889
Blond 0.7401575 0.05511811 0.5409091 0.11818182
Brown 0.2937063 0.41608392 0.1971831 0.19718310
Red   0.2394366 0.36619718 0.4531250 0.21875000

> rowSums(aaply(HE, 1, percents))
Black Blond Brown   Red
1     1     1     1

> aaply(HE, 2, percents)
Eye          Black      Blond     Brown        Red
Blue  0.09302326 0.43720930 0.3906977 0.07906977
Brown 0.30909091 0.03181818 0.5409091 0.11818182
Green 0.07812500 0.25000000 0.4531250 0.21875000
Hazel 0.16129032 0.10752688 0.5806452 0.15053763

> rowSums(aaply(HE, 2, percents))
Blue Brown Green Hazel
1     1     1     1
1. Extract a subset of the data for which it is easy to solve the problem
2. Solve the problem by hand, checking as you go
3. Write a function that encapsulates the solution
4. Use appropriate **ply function to
   - split the data into pieces,
   - apply the function to each piece
   - join the pieces back together
Example: baseball, 1871-2007

Batting records of all US players, 1871-2007 with 15+ years of data

Focus on year, rbi (runs batted in), ab (at bats): performance over career
# how many unique players?
> length(unique(baseball$id))
[1] 1228
# examine career trajectory of baseball players in terms of rbi/ab
# look at one player: Babe Ruth
> baberuth <- subset(baseball, id == "ruthba01")
> baberuth <- transform(baberuth, cyear= year - min(year) + 1)

> plot(rbi/ab ~ cyear, data=baberuth, type='b')
> BRmodel <- lm(rbi/ab ~ cyear, data=baberuth)
> abline(BRmodel, col="red")
# apply transform() for all players
baseball <- ddply(baseball, .(id), transform, cyear = year - min(year) + 1)

define plot function for one player, with common scale:

```r
xlim <- range(baseball$cyear, na.rm=TRUE)
ylim <- range(baseball$rbi/baseball$ab, na.rm=TRUE)
plotfun <- function(df) {
  plot(rbi/ab ~ cyear, data=df, type='b', xlim=xlim, ylim=ylim)
  abline(lm(rbi/ab ~ cyear, data=df, col="red"))
}
```

use `d_ply()`: make plots for all 1128 players:

```r
pdf("bbplots.pdf", width=8, height=4)
d_ply(baseball, .(reorder(id, rbi/abi)), plotfun)
dev.off()
```
# restrict ourselves to players with > 25 at bats
> bb <- subset(baseball, ab >= 25)
> length(unique(bb$id))
[1] 1152

Fit models: linear

> # function to fit one model
> model1 <- function(df) {
+   lm(rbi / ab ~ cyear, data=df)
+ }
> model1(baberuth)

Call: 
  lm(formula = rbi/ab ~ cyear, data = df)

Coefficients: 
(Intercept)        cyear
           0.203200     0.003413

> # apply to all
> model1s <- dlply(bb, .(id), model1)

Fit models: quadratic

> # or, try a quadratic model
> model2 <- function(df) {
+   lm(rbi / ab ~ cyear + I(cyear^2), data=df)
+ }
> model2(baberuth)

Call: 
  lm(formula = rbi/ab ~ cyear + I(cyear^2), data = df)

Coefficients: 
(Intercept)        cyear   I(cyear^2)
          0.1267872 0.0225163 -0.0008306

> # apply to all
> model2s <- dlply(bb, .(id), model2)
We have a list of 1152 models, one for each player: summarize them:

```r
> # extract an R^2 from a model
> rsq <- function(x) summary(x)$r.squared
>
> # summarize all
> summaries <- ldply(model1s, function(x) c(coef(x), rsquare = rsq(x)))
> names(summaries)[2:3] <- c("intercept", "slope")
> head(summaries, 4)

    id         intercept        slope     rsquare
   1 aaronha01 0.18329371  0.0001478121 0.000862425
   2 abernte02 0.00000000            NA 0.000000000
   3 adairje01 0.08670449 -0.0007118756 0.010230121
   4 adamsba01 0.05905337  0.0012002168 0.030184694

> plot(density(summaries$rsquare, na.rm=TRUE), xlab="R^2", main="")
> rug(summaries$rsquare, col="red")
```
Where to go from here?

• This account (n=1) entirely impressionistic
  ▪ Some features of programming languages
  ▪ Characteristics: power, elegance, suggestivity, generality, economy, ...

• How to study empirically?
  ▪ Experiment: tasks? population? language features? what needs to be controlled?
  ▪ Survey: population? language features? what needs to be controlled?