Statistical Computing (36-350)

Lecture 6: Top-Down Design

Cosma Shalizi

18 September 2012

Agenda

- Top-down design of programs
- Example: Linear regression in R

ESSENTIAL READING FOR FRIDAY: Sec. 7.6 and 7.9 of the textbook

Top-Down Design

- Start with the big-picture view of the problem
- Break the problem into a few big parts
- Figure out how to fit the parts together
- Go do this for each part

The Big-Picture View

Resources: what information is available as part of the problem? (Usually arguments)
Requirements: what information do we want as part of the solution? (Usually return values)
What do we have to do to transform the problem statement into a solution?

Breaking Into Parts

Try to break the calculation into a *few* (say ≤ 5) parts

- Bad: write 500 lines of code, chop it into five 100-line blocks
- Good: each part is an independent calculation, using separate data

Advantages of the good way:

- More comprehensible to human beings
- Easier to improve and extend (respect interfaces)

Put the Parts Together

Assume that you can solve each part, and their solutions are functions
Write top-level code for the function which puts those steps together:

```
# Not actual code
big.job <- function(lots.of.arguments) {
   intermediate.result <- first.step(some.of.the.args)
   final.result <- second.step(intermediate.result,rest.of.the.args)
   return(final.result)
}
```

The sub-functions don't have to be written when you *declare* the main function, just when you *run* it

What About the Sub-Functions?

Recursion: Because each sub-function solves a single well-defined problem, we can solve it by top-down design

The step above tells you what the arguments are, and what the return value must be (interface)

The step above doesn't care how you turn inputs to output (internals)

Stop when we hit a sub-problem we can solve in a few steps with *built-in* functions

credit: `http://cheezburger.com/View/4517375744`

Thinking Algorithmically

Top-down design only works if you understand

- the problem, and
- a systematic method for solving the problem

∴ it forces you to think **algorithmically**

First guesses about how to break down the problem are often wrong

but functional approach contains effects of changes

∴ don't be afraid to change the design

Example: Coding up linear regression

Basic form of the model:

$$Y = X^T \beta + \text{noise}$$

Least-squares estimation: find $\hat{\beta}$ such that the mean squared error $n^{-1} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2$ is minimized
Solution:

$$\hat{\beta} = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$$

where $\mathbf{x}$ is $n \times p$ matrix of inputs, $\mathbf{y}$ is $n \times 1$ matrix of responses
$n^{-1}(\mathbf{x}^T \mathbf{x})$ is covariance matrix of X
$n^{-1}\mathbf{x}^T \mathbf{y}$ is vector of covariances of X and Y

How R Does This

The `lm` (linear model) function:

```
lm(formula,data, [[many other options]])
```

`formula`: something like `Hwt ~ 1 + Sex + Bwt + Bwt:Sex`
`data`: Data frame to look for the variables in the formula in
Return value is an `lm` object, with coefficients, standard errors, residuals, ...
How would we write this?

What's the top level of the design?

1. Prepare design matrix **x** from formula and data
2. Prepare response vector **y** from formula and data
3. Calculate coefficients from **x** and **y**
4. Prepare extra information (residuals, standard errors, etc.) from **x**, **y** and coefficients
5. Return everything

Now go do each of the top-level steps (in any order)

Calculating the Coefficients

Remember the OLS estimator:

$$\hat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

This is a one-liner in R:

```
beta.hat = solve(t(X) %*% X) %*% t(X) %*% Y
```

(remember `solve(A)` returns A^{-1})

but what if $\mathbf{x}^T\mathbf{x}$ is singular?

Making the Design Matrix

Roughly, each term on the RHS of the formula leads to a column in $\mathbf{X}$

1. Plain variables (like `Bwt`) need a column each
2. An intercept needs a column of 1s
3. Dummy or indicator variables need binary columns (like `Sex`)
4. Interactions need a column of products (like `Bwt:Sex`)
5. Transformations (say `log`)

Use names on these columns to match formula terms, so these carry through to the coefficients
Making $\mathbf{Y}$ is similar $\therefore$ lots of common sub-sub-functions
...

Summary

1. Top-down design is a recursive heuristic for coding
    1. Split your problem into a few sub-problems; write code tying their solutions together
    2. If any sub-problems still need solving, go write their functions
2. Leads to many short functions, each solving one well-defined problem
3. Disciplines you to think algorithmically