Lecture 2
R in Modeling and Optimization

BIOST 2094 Spring 2017
Tianzhou Ma (Charles)

Department of Biostatistics
School of Public Health
University of Pittsburgh

January 13th, 2017

Table of contents

Summary Statistics - Vectors

- Functions for calculating summary statistics of vector elements

mean(x)	Mean of x
median(x)	Median of x
var(x)	Variance of x
sd(x)	Standard deviation of x
cov(x,y)	Covariance of x and y
cor(x,y)	Correlation of x and y
min(x)	Minimum of x
max(x)	Maximum of x
range(x)	Range of x
quantile(x)	Quantiles of x for the given probabilities

Let's quickly do some summary statistics for a randomly generated vector:

```
set.seed(15213)
x <- rnorm(100)
mean(x);var(x);range(x);quantile(x)

## [1] -0.06482466
## [1] 1.237393
## [1] -2.886257  2.369018
##          0%         25%         50%         75%        100%
## -2.88625745 -0.72789139 -0.08828617  0.66585515  2.36901755
```

Summary Statistics - Dataframes

- Functions for calculating summary statistics of the columns of a dataframe

summary()	Summary statistics of each column; type of statistics depends on data type
apply()	Apply a function to each column, works best if all columns are the same data type
tapply()	Divide the data into subsets and apply a function to each subset, returns an array
by()	Similar to tapply(), return an object of class by
ave()	Similar to tapply(), returns a vector the same length as the argument vector
aggregate()	Similar to tapply(), returns a dataframe
sweep()	"Sweep out" a summary statistic from a dataframe, matrix or array

```r
#state.x77 is a built-in matrix of some facts and figures of 50 US states
area.factor <- cut(state.x77[,"Area"],breaks=quantile(state.x77[,"Area"],
                   c(0,.25,.75,1)),labels = c("small","medium","large"),
                   include.lowest= TRUE)
state <- data.frame(pop = state.x77[,"Population"],
                    inc = state.x77[,"Income"],
                    area = area.factor,
                    region = state.region)
head(state,3);dim(state)
```

```
##            pop  inc   area region
## Alabama   3615 3624 medium  South
## Alaska     365 6315  large   West
## Arizona   2212 4530  large   West
## [1] 50  4
```

```r
args(tapply);args(by)

## function (X, INDEX, FUN = NULL, ..., simplify = TRUE)
## NULL
## function (data, INDICES, FUN, ..., simplify = TRUE)
## NULL

with(state, tapply(inc, list(region, area), FUN=mean))

##                small   medium    large
## Northeast       4540 4676.000       NA
## South           4340 3876.636 4188.000
## North Central   4458 4620.600 4669.000
## West            4963 4864.000 4664.273

with(state, tapply(inc, list(region, area), FUN=var))

##                   small    medium    large
## Northeast     394784.7 103058.00       NA
## South         719798.7 267629.05       NA
## North Central       NA  94866.49       NA
## West                NA        NA 517914.8
```

Try out "sweep":

```
#sweep out the column means for population and income,
#i.e. substract the overall mean
(col.mean <- apply(state[,1:2],2, mean))

##     pop     inc
## 4246.42 4435.80

args(sweep)

## function (x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)
## NULL

sweep(state[1:5,1:2],2, col.mean, FUN="-")

##                pop     inc
## Alabama     -631.42  -811.8
## Alaska     -3881.42  1879.2
## Arizona    -2034.42    94.2
## Arkansas   -2136.42 -1057.8
## California 16951.58   678.2
```

Apply family and related functions

apply()	Apply a function to the margins of an array
lapply()	Apply a function to each element of a list or vector, returns a list
sapply()	Same as lapply(), but returns a vector or matrix by default
vapply()	Similar to sapply(), but has a pre-specified type of return value
mapply()	Multivariate version of sapply(), apply a function to the corresponding elements from multiple lists or vectors
tapply()	Apply a function to each subset of a vector, where the subsets are determined by the unique combination of factor levels
outer()	General outer product of two vectors
sweep()	Sweep out a summary statistic from the margins of an array
replicate()	Repeated evaluation of an expression

- These functions allow code to be more compact and clearer

Contingency Tables

- The following functions are used for creating data tables

table()	Create a contingency table of the counts at each combination of factor levels
ftable()	Similar to table(), useful for multidimensional tables
xtabs()	Create a contingency table using a formula interface, useful if the data have already been tabulated also includes an argument for specifying a dataframe
prop.table()	Table of proportions, relative to the given margin
addmargins()	Add margins to a table (default is to sum over all margins in the table)

- Although it appears that these functions return matrices or arrays, these functions actually return objects of class "table" for table(), "ftable" for ftable(), and "xtabs" for xtabs().

```
state$pop.cut <- with(state, cut(pop,
                  breaks=c(min(pop),median(pop),max(pop)),
                   labels = c("small", "large"),
                    include.lowest = T) )
with(state, ftable(pop.cut,area,region))
```

```
##                region Northeast South North Central West
## pop.cut area
## small   small                 4     3               0    1
##         medium                0     3               3    0
##         large                 0     0               1   10
## large   small                 3     1               1    0
##         medium                2     8               7    1
##         large                 0     1               0    1
```

Functions to conduct statistical test

t.test()	Two-sample t-test
wilcox.test()	Wilcoxon rank-sum test
power.t.test()	Compute the power for a t-test
var.test()	F-test to compare variances of two samples (normality)
prop.test()	Compare proportions using a normal approximation
binom.test()	Perform an exact binomial test
power.prop.test()	Compute the power for a two-sample test for proportions
shapiro.test()	Shapiro-Wilk test of normality
cor.test()	Test for correlation/association between paired samples
chisq.test()	Perform a chi-square test of independence
fisher.test()	Perform Fisher's exact test

Fit Linear Models

- Functions for fitting linear models
  - `lm()` Fits linear models (linear regression or ANOVA)
  - `aov()` Fits *balanced* ANOVA model; returns Type I, sequential sum of squares
- Main difference between lm() and aov() is the way summary() handles the results. The summary table for aov() is one row for each categorical variable and the summary table for lm() has one row for each estimated parameters (i.e. one row for each factor level)
- Basic syntax for lm() (similar syntax for aov()),

```
lm(formula, data)
```

  - `formula` Symbolic description of the model
  - `data` Optional dataframe containing the variables in the model
- summary.lm() and summary.aov() summarize a linear model and ANOVA model, respectively

Formulas

- Basic form of a formula,

$$\texttt{response} \sim \texttt{model}$$

- Formula notation,

‘ + ’	Separates main effects
‘ : ’	Denotes interactions
‘ * ’	All main effects and interactions
‘^n’	Include all main effects and n-order interactions
‘ - ’	Removes the specified terms
‘ \ ’	Nested effects
I( )	Brackets the portions of a formula where operators are used mathematically
‘ . ’	Main effect for each column in the dataframe, except the response

Formulas

- Sample formulas, for a model with response y and predictors a, b and c

Model	**Interpretation**
y∼1	Just the intercept
y∼a	One main effect
y∼-1+a	No intercept
y∼a+b	Two main effects
y∼a+b+c+a:b	Three main effects and an interaction between a and b
y∼a*b	All main effects and interactions (same as a+b+a:b)
y∼factor(a)	Create dummy variables for a (if not already a factor)
y∼(a+b+c)^2	All main effects and second-order interactions
y∼(a+b+c)^2 - a:b	All main effects and second-order interactions except a:b
y∼I(a^2)	Transform a to a^2
log(y)∼a	Log transform y
y∼a/b/c	Factor c nested within factor b within factor a
y∼.	Main effect for each column in the dataframe

Inference for Linear Models

- Functions used for performing inference

anova()	Compute an ANOVA table for model terms or compare nested models; returns Type I, sequential sum of squares
drop1()	Test factors using the Type III, marginal sum of squares
confint()	Confidence intervals for model parameters
predict.lm()	Get the average response value for predictors included and not included in the model; get confidence and prediction intervals for the fitted values
TukeyHSD()	Multiple comparisons, Tukey's Honest Significant Difference
pairwise.t.test()	Pairwise t-tests, correcting for multiple comparisons

Model Diagnostics

- Several functions provide information used with model diagnostics

fitted.values()	Returns fitted values
residuals()	Returns residuals
rstandard()	Standardized residuals, variance one; residual standardized using overall error variance
rstudent()	Studentized residuals, variance one; residual standardized using leave-one-out measure of the error variance
qqnorm()	Normal quantile plot
qqline()	Add a line to the normal quantile plot
plot.lm()	Given a lm object produces six diagnostic plots, selected using the which argument; default is plots 1-3 and 5 1 Residual versus fitted values 2 Normal quantile-quantile plot 3 $\sqrt{\lvert \textit{Standardized residuals} \rvert}$ versus fitted values 4 Cook's distance versus row labels 5 Standardized residuals versus leverage along with contours of Cook's distance 6 Cook's distance versus leverage/(1-leverage) with $\sqrt{\lvert \textit{Standardized residuals} \rvert}$ contours

Model Selection

- Functions for model selection

step()	Choose a model by AIC in a stepwise algorithm
AIC()	Compute the AIC for the fitted model
BIC()	Compute the BIC for the fitted model
anova()	Given multiple models tests the models against one another in the order specified
add1()	Add one term to a model and compute the change in fit
drop1()	Drop one term from a model and compute the change in fit

Generalized Linear Models

- Generalized Linear Models are fit using the function glm(). Basic syntax,

```
glm(formula, family = gaussian, data)
```

- The family argument specifies the error distribution and link function. See ?family for more information

```
binomial(link = "logit")
gaussian(link = "identity")
poisson(link = "log")
```

- Almost all of the functions discussed previously that work with lm objects have corresponding methods for glm objects. Or are generic enough that they apply to both lm objects and glm objects. For example,

summary.glm()	Summarize the model fit
anova.glm()	Analysis of deviance table
confint.glm()	Confidence interval for model parameters
predict.glm()	Obtain predicted values
influence.measures()	Measures of influence
step()	Step-wise selection using AIC
drop1()	Test parameter using deviance

Survival Analysis

- The survival package comes with R but still needs to be loaded before you can use the functions.
- For an overview of other R packages available for survival analysis see, `http://cran.rproject.org/web/views/Survival.html`
- Almost all survival analysis functions use a survival object created by Surv() that consist of the event time and event indicator
- Basic syntax,

```
Surv(time, event)
```

time	For right censored data, follow-up time
event	Event indicator default is 1=event and 0=censor, for a different event value use '=='

Survival Analysis

- Functions useful for survival analysis

survfit.formula()	Kaplan-Meier estimate
survfit.coxph()	Predicted survival curve from a Cox model
survdiff()	Log-rank and Harrington and Fleming weighted log-rank test; $w(t) = \hat{S}(t)^{\rho}, \rho = 0$ for log-rank test
survreg()	Parametric Proportional Hazards Model
coxph()	Cox proportional hazards model
cox.zph()	Tests the proportional hazards assumption
summary()	Summarize results
anova.coxph()	Analysis of deviance table for one or more Cox models
confint()	Confidence intervals of parameter estimates
drop1()	Test each factor individually
step()	Stepwise algorithm using the AIC
plot.survfit()	Plot a survival curve

- Remember the function methods() is very useful for finding the methods that correspond to a generic function or the methods for a particular class

Mixed Models

- Subject-specific or cluster-specific model of correlated/clustered data
- Basic premise is that there is natural heterogeneity across individuals in the study population that is the result of unobserved covariates; random effects account for the unobserved covariates.

- The lme4 package contains functions for fitting linear mixed models, generalized linear mixed models and nonlinear mixed models

- The lme4 package uses S4 classes and methods.
  - Information in S4 classes is organized into slots. Each slot is named and requires a specified class.
  - Use the @ to extract information from a slot.
  - To get the names of the slots use, getSlots("*class name*")
- For more information about fitting mixed models in R using lme4 see the available vignettes, vignette(package="lme4")

lmer()

- The lmer() function in the lme4 package is used to fit linear and generalized linear models.
- Basic syntax,

```
lmer(formula, data, family=NULL, REML=TRUE)
```

formula	Symbolic description of the model to be fitted
data	Optional dataframe
family	Description of the error distribution and link function, if NULL a linear mixed model is fitted
REML	Logical, if TRUE estimate using REML (provides a consistent estimate of the variance components); if FALSE estimate using ML

Formula lmer()

- A random-effects term in lmer() is specified by a linear model term and a grouping factor separated by '|'; i.e. a random effect is a linear model term conditional on the level of the grouping factor.
- The entire random-effects expression should be enclosed in parentheses since the precedence of '|' as an operator is lower than most other operators used in linear model formulas
- For example,
  - Random intercept,
    `lmer(Reaction ~ Days + (1 | Subject), data=sleepstudy)`
  - Random intercept and slope,
    `lmer(Reaction ~ Days + (Days | Subject), data=sleepstudy)`
- See the vignettes for how to fit nested random effects, vignette(package="lme4")

Inference

- Functions used for inference and prediction,

summary()	Summarize model results
anova()	Sequential tests of fixed effects and model comparison
VarCorr()	Extract variance components
ranef()	Predict random effects
residuals()	Extract residuals

Generalized Estimating Equations (GEE)

- Population-average or marginal model, provides a regression approach for generalized linear models when the responses are not independent (correlated/clustered data)
- Goal is to make inferences about the population, accounting for the within-subject correlation
- The packages gee and geepack are used for GEE models in R
- The major difference between gee and geepack is that geepack contains an ANOVA method that allows us to compare models and perform Wald tests.

Generalized Estimating Equations

- Basic Syntax for geeglm() from the geepack package; has a syntax very similar to glm()

```
geeglm(formula, family=gaussian, data, id, zcor=NULL, constr,
                                    std.err="san.se")
```

formula	Symbolic description of the model to be fitted
family	Description of the error distribution and link function
data	Optional dataframe
id	Vector that identifies the clusters
zcor	Enter a user defined correlation structure
constr	Working correlation structure: "independence", "exchangeable", "ar1", "unstructured", "userdefined"
std.err	Type of standard error to be calculated. Default "san.se" is the robust (sandwich) estimate; use "jack" for approximate jackknife variance estimate

Correlation Structure

- Independence,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Exchangeable,

$$\begin{pmatrix} 1 & \rho & \rho \\ \rho & 1 & \rho \\ \rho & \rho & 1 \end{pmatrix}$$

- Autoregressive order 1,

$$\begin{pmatrix} 1 & \rho & \rho^2 \\ \rho & 1 & \rho \\ \rho^2 & \rho & 1 \end{pmatrix}$$

- Unstructured,

$$\begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{12} & 1 & \rho_{23} \\ \rho_{13} & \rho_{23} & 1 \end{pmatrix}$$

- GEE model will give valid results with a misspecified correlation structure when the sandwich variance estimator is used

Inference

- For a geeglm object returned by geeglm(), the functions drop1(), confint() and step() do not apply; however anova() does apply.
- The function esticon() in the doBy package computes and test linear functions of the regression parameters for lm, glm and geeglm objects
- Basic syntax,

```
esticon(obj, cm, beta0, joint.test=FALSE)
```

obj	Model object
cm	Matrix specifying linear functions of the regression parameters (one linear function per row and one column for each parameter)
beta0	Vector of numbers
joint.test	If TRUE joint Wald test of the hypothesis Lbeta=beta0 is made, default is one test for each row, (Lbeta).i=beta0.i

esticon()

- Let $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)$ denote the estimated parameters. Also let $k = (k_1, \ldots, k_p)$ denote a vector of constants; one row of the matrix for the cm argument. Then $c = k^T \beta = k_1\beta_1 + \ldots + k_p\beta_p$.
- esticon() calculates the linear combinations of the parameter estimates c, the standard error and the confidence interval
- Specify a value for beta0 to test $H_0 : c = beta0$
- If `joint.test=TRUE` then all of the linear combinations are tested jointly

Optimization

- In mathematics, "optimization" or "mathematical programming" refers to the selection of a best element (with regard to some criterion) from some set of available alternatives.
- A typical optimization problem usually consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.
- "Convex programming" studies the case when the objective function is convex (minimization) or concave (maximization) and the constraint set is convex.

Optimization

- R has several functions for optimization,

optimize()	One dimensional optimization, no gradient or Hessian
optim()	General purpose optimization, five possible methods, gradient optional
constrOptim()	Minimization of a function subject to linear inequality constraints, gradient optional
nlm()	Non-linear minimization, can optionally include the gradient and hessian of the function as attributes of the objective function
nlminb()	Minimization using PORT routines, can optionally include the gradient and Hessian of the objective function as additional arguments

- These functions use different algorithms and accept different arguments, no one function is superior to the others. Which function you use depends on your particular problem; use optimize() for one-dimensional problems.
- To turn a minimization problem into a maximization problem, multiply the objective function and gradient by -1.
- There are also packages with additional optimization functions, http://cran.r-project.org/web/views/Optimization.html

	LP	MILP	SOCP	MISOCP	SDP	GP	NLP	MINLP	R	Matlab	Julia	Python	Cost
modeling tools													
JuMP.jl	x	x	x	x			x	x			x		O
Convex.jl	x	x	x	x	x						x		O
cvx	x	x	x	x	x	x				x		x	A
convex solvers													
Gurobi	x	x	x	x					x	x	x	x	A
Mosek	x	x	x	x	x	x	x		x	x	x	x	A
CPLEX	x	x	x	x					?	x	x	x	A
SCS	x		x		x					x	x	x	O
SeDuMi	x		x		x	?				x			O
SDPT3	x		x		x	?				x			O
NLP solvers													
KNITRO	x	x					x	x	x	x	x	x	$
NLopt	x						x			x	x	x	O
Ipopt	x						x			x	x	x	O

- O: open source
- A: free academic license
- $: commercial

One-Dimensional Optimization

```
optimize(f, interval, ..., maximum=FALSE)
```

f	Function to be optimized
interval	Vector giving the interval c(lower, upper) to be searched
...	Additional arguments to be passed to f
maximum	Logical, find maximum if TRUE

- Cannot specify the gradient to assist with the optimization

Univariate Optimization

Practice

- Use the optimize() function to find all of the local maximum and minimums of the function,

$$f(x) = \begin{cases} 0 & x = 0 \\ |x| log(|x|/2) e^{-|x|} & x \neq 0 \end{cases}$$

Plot the function to get an idea of where to look.

Multi-Dimensional Optimization

```
optim(par, fn, gr=NULL, ..., method, control)
```

par	Initial values
fn	Function to be optimized, argument is a vector of parameters
gr	A function that returns the gradient, same argument as fn
...	Additional arguments passed to fn
method	Method to be used
control	List of control parameters (number of iterations, tolerance, etc.)

- By default the minimum is found, to find the maximum set the control parameter fnscale to -1, control=list(fnscale=-1). This divides the objective function and gradient by -1.

Constrained Optimization

- For box constraints use the "L-BFGS-B" method in optim() and the arguments lower and upper to give bounds for the arguments
- For linear inequality constraints use constrOptim()

```
constrOptim(theta, f, grad, ui, ci, control, ...)
```

theta	Starting values, $px1$ vector
f	Function to be optimized
grad	Function that returns the gradient
ui	Constraint matrix, kxp
ci	Constraint vector, $kx1$ vector
control	List of control parameters
...	Additional arguments passed to f

- Feasible region is defined by ui %*% theta - ci $>= 0$

Write your own optimization function

- Let's try to implement the Gradient descent algorithm.
- Consider an unconstrained, smooth convex optimization problem: min $f(x)$, and let's assume f is convex and differentiable with $dom(f) = R^n$.
- The Gradient descent works as follows:
  Choose an initial $x^{(0)} \in R^n$, and repeat:
  $x^{(k)} = x^{(k-1)} - t_k \nabla f(x^{(k-1)})$, for $k = 1, 2, \ldots$, stop when converged.
- The idea comes from the following expansion:

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} ||y - x||_2^2$$

  where the usual $\nabla^2 f(x)$ term is replaced by $\frac{1}{t} I$, and we choose the next point x^+ to minimize this approximation $x^+ = x - t \nabla f(x)$.
- The step size t_k can be chosen in a smart way, but here for simplicity, we assume a fixed step size.