R

Simulation and Optimization

R.M. Ripley

Department of Statistics
University of Oxford

2008/9

Uses of Simulation

- Statistical models are often mathematically intractable.
- Generate multiple samples from model by simulation.
- Use these samples to investigate the behaviour of the model.
- Need realizations of random variables with various different distributions, our ***random numbers***.
- Details of how to use the random numbers to create samples will not be considered here.

Random number generation

- Random numbers calculated on a computer are not random.
- They are **pseudo-random**, following a predicted sequence, but in the short-term (i.e. anything but *very* long-term) will appear random.
- This is useful, as two sets of random numbers of the same size from the same generator using the same initial value will be exactly the same.
- Details of generation of random samples from standard distributions need not concern us, as R provides functions.

Using Random Number Generators in R

- Each time a random number is required, R will use and update a variable called **`.Random.seed`** which is in your workspace.
- At the first use, if **`.Random.seed`** does not exist, one will be created, with a value generated from the time.
- The function **`set.seed(n)`** will set **`.Random.seed`** to a value derived from the argument **`n`**. Use **`set.seed()`** to repeat the same sequence of random numbers.
- It is possible to save and restore **`.Random.seed`** within your functions, but take care with *scope* (see (much) later!).

R functions for generating random samples

To obtain a sample of size **`n`**

`runif(n,min=0,max=1)`	Uniform
`rnorm(n,mean=0,sd=1)`	Normal
`rexp(n,rate=1)`	Exponential with mean 1/rate
`rt(n,df)`	t with df degrees of freedom
`rbinom(n,size,prob)`	Binomial, successes in **`size`** trials with probability of success **`prob`**

The parameters can usually be specified as vectors, so that non-iid samples can be obtained. For example,
`contam<-rnorm(100,0,(1+2*rbinom(100,1,0.05)))`
will generate 100 samples from the contaminated normal distribution in which a sample is from $N(0,1)$ with probability 0.95 and otherwise from $N(0,9)$.

Further sampling

The function **`sample`** re-samples from a data vector, with or without replacement. It has several forms:

`sample(n)`	select a random perm. from $1,\ldots,n$
`sample(x)`	randomly permute **`x`**, for **`length(x)>1`**
`sample(x,repl=T)`	a bootstrap sample
`sample(x,n)`	sample **`n`** items from **`x`** w/o replacement
`sample(x,n,repl=T)`	sample **`n`** items from **`x`** with replacement
`sample(x,n, repl=T,prob=probs)`	probability sample of **`n`** items from **`x`**.

The Optimization Problem

- Given a function **`f(x)`**, what value of **`x`** makes **`f(x)`** as small or as large as possible?
- In a statistical context, **`x`** will usually be the ***parameters*** of a model, and **`f(x)`** either the model likelihood to be maximized or some measure of discrepancy between data and predictions to be minimized
- The optimal set of parameters will give the *best fit*.
- Only need to consider *small* as **`-f(x)`** is *large* when **`f(x)`** is *small*.
- We consider here ***general-purpose optimizers***.

Local and Global Minima

- The (negative of the) likelihood for the General Linear Model (and that for many other linear models) is well-behaved: it has a single, global minimum.
- For more complicated models there may be many local minima.
- Finding a global minimum is difficult, and not always important. Only if local minima are widely separated in parameter space are they likely to invalidate our conclusions.
- We will concentrate on methods of finding local minima.
- Check for different local minima by altering the initial values, algorithm used, or other parameters of the fitting process.

Univariate Optimization

optimize (or **optimise**) finds a (possibly local) minimum of a function in a specified interval with respect to its first argument.

- Function to be minimized is first argument of **optimize**.
- Can pre-specify the function or include it in the command (cf. panel functions)

Example

```
f <- function (x,a) (x-a)^2
xmin<- optimize(f, interval=c(0, 1),  a = 1/3)
or
xmin<- optimize(function(x,a)(x-a)^2,
          interval= c(0, 1),  a = 1/3)
```

Note how the (fixed) parameter **a** is passed into **f**.

Other optimize() Arguments

- Interval within which to search can be specified by **interval=** or **upper=, lower=**.
- To *maximize*, set **maximum=TRUE**.
- Accuracy can be set using the **tol=** argument.
- Note the order of arguments: **optimize(f,interval,...,lower,upper,maximum,tol)**
- The **...** can be named or unnamed and will be passed to **f**.
- Arguments after the **...** must be specified by name.
- **optimize** returns a list with two items:

minimum	The value of **x** at which **f(x)** is minimized.
objective	The value of **f(x)** at **x=minimum**

General Optimization

- In more than one dimension the problem is harder.
- R has several different functions: most flexible is **optim()** which includes several different algorithms.
- Algorithm of choice depends on how easy it is to calculate derivatives for the function. Usually better to supply a function to calculate derivatives, but may be unnecessary extra work.
- Ensure the problem is scaled so that unit change in any parameter gives approximately unit change in objective.
- Experiment with different methods...

optim(): Nelder-Mead method

- The **default** method
- Basic idea: for a function with **n** vertices, choose a polygon with **n+1** vertices. At each step, alter vertex with minimum **f(x)** to improve the objective function, by *reflection*, *expansion* or *contraction*
- Does not use derivative information
- Useful for non-differentiable functions
- May be rather slow

optim(): BFGS method

- A **quasi-Newton** method: builds up approximation to Hessian matrix from gradients at start and finish of steps
- Uses the approximation to choose new search direction
- Performs line search in this direction
- Update term for the Hessian approximation is due to Broyden, Fletcher, Goldfarb and Shanno (proposed separately by all four in 1970)
- Uses derivative information, calculated either from a user-supplied function or by finite differences
- If dimension is large, the matrix stored may be very large

optim(): CG method

- A **conjugate gradient** method: chooses successive search directions that are analogous to axes of an ellipse
- Does not store a Hessian matrix
- Three different formulae for the search directions are implemented: *Fletcher-Reeves, Polak-Ribiere* or *Beale-Sorenson*
- Less robust than BFGS method
- Uses derivative information, calculated either from a user-supplied function or by finite differences

optim(): L-BFGS-B method

- **A limited memory version of BFGS**
- Does not store a Hessian matrix, only a limited number of update steps for it
- Uses derivative information, calculated either from a user-supplied function or by finite differences
- Can restrict the solution to lie within a "box", the only method of `optim()` that can do this

optim(): SANN method

- A variant of **simulated annealing**
- A stochastic algorithm
- Accepts changes which increase the objective with positive probability (when minimising!)
- Does not use derivative information
- Can be very slow to converge but may find a ‘good’ solution quickly

How to use optim()

```
optim(par,fn,gr=NULL,...,
method=c("Nelder-Mead","BFGS","CG","L-BFGS-B","SANN"),
lower=-Inf,upper=Inf,control=list(),hessian=FALSE)
```

par	Starting values of the parameters.
fn	The function (supply as for `optimize`)
gr	Function to calculate the derivative. Only relevant for methods “BFGS”, “CG”, or “L-BFGS-B”.
...	Other parameters for (both) `fn` and `gr`.
method	Algorithm to use
lower,upper	Vectors of limits for parameters if required. Only allowed if `method='L-BFGS-B'`.
control	control options: see next slide
hessian	Logical: whether to return a hessian estimate calculated by finite differences.

optim(): Control Options

There are very many. The most important are:

trace	A positive integer: higher values give more information
fnscale	An overall scaling: if negative, maximization will be performed
parscale	A vector of scalings for the parameters
maxit	Maximum number of iterations to be performed: may be useful to terminate unsuccessful attempts Also used to perform one or two steps if convergence is unimportant
type	Used to select formula for “CG” method

optim(): Components of Return Value

par	best set of parameters
value	value of `fn` corresponding to `par`
counts	number of calls to `fn` and `gr`: excludes calls for purposes of approximating derivatives or Hessian
convergence	error code: 0=success, 1=max iterations reached, 10= degeneracy of Nelder-Mead simplex, 51=warning from “L-BFGS-B”, 52=error from “L-BFGS-B”
message	further information, if any
hessian	If requested, a symmetric matrix estimate of the Hessian at the solution: Hessian of unconstrained problem, even if constraints are active

Constrained Optimization

- Often possible to constrain parameters to be positive by transforming to logarithmic scale
- `optim()` with method=“L-BFGS-B” will accept box constraints
- `nlminb` is an alternative function allowing box constraints
- `constrOptim` is a wrapper for `optim` which enforces linear inequality constraints by adding a barrier function: can use with any `optim` method except “L-BFGS-B”

Arguments of constrOptim

theta	starting values of parameters
f	function
grad	gradient. Must be given if using a method which needs derivatives.
ui, ci	constraints are that **`ui %*% theta - ci>= 0`**
mu	tuning parameter which takes small values
control	as for **`optim`**
method	as for **`optim`**
outer.iterations	Number of iterations in outer loop
outer.eps	Accuracy required in outer loop
...	other parameters for your function

Returns values as for **`optim`** plus value of barrier function and number of outer iterations

Exercises 5 Page 1

- In the package **`survival`** there is a dataset called **`ovarian`**. Your task is to fit a log normal model to relate survival time to age, and treatment, using **`optim`**.
- The likelihood to be used is

$$\prod_{\delta_i=1} f(t_i) \prod_{\delta_i=0} (1 - F(t_i))$$

  where δ_i is the censoring indicator, **`ovarian$fustat`**, $f(t_i)$ is the density and $F(t_i)$ the cumulative distribution. The mean to be fitted, on a log scale, will be $\mathbf{x}\beta$ where $\mathbf{x}$ is the covariate matrix with a column of 1's added for the mean, and β are parameters to be estimated. A common variance should be estimated.

Exercises 5 Page 2

- Write a function to calculate the likelihood from the parameters, and maximize it using **`optim`**. Then write a function to calculate the derivatives and use that as well. Try different methods of fitting.
- Hints:
  - Transform σ to a log scale to avoid constraints.
  - Start with just the mean and no covariates.
  - For **`method='BFGS'`**, you may need to use **`parscale`**. If you know the approximate gradients, set **`parscale`** proportional to these. Even if not essential it may speed things up.
  - After writing the function to calculate the derivatives, check it using finite differences and the function which calculates the likelihood.
- Check your results using **`survreg`**.