R computing for Business Data Analytics

Instructor: Dr. Howard Hao-Chun Chuang

Assistant Professor, Department of Management Information Systems

National Chengchi University, Taipei, Taiwan 116

chuang@nccu.edu.tw

Last Revised: November 2014

A good data analyst is expected to have a solid grasp of numerical methods. Lecture 9 will cover various functions in *R* for you to tackle *calculus* and *optimization*. The content of this lecture is largely adapted from chapters 7 & 8 in Bloomfield (2014).

9.1 Differentiation & integration

Consider the function below

$$f(x) = x^3 \sin(x/3)\log(\sqrt{x})$$

```
> f=expression(x^3*sin(x/3)*log(sqrt(x)))
> g=D(f, “x”)
> gfun=function(x){eval(g)}
```

Suppose x=x0=1

```
> x0=1
> gfun(x0)
> exact=gfun(x0)
```

This will give us the *exact* value of differentiation. Nevertheless, people sometimes just use numerical approximations. From calculus we know

$$\frac{df(x)}{dx} = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

Hence, one way to calculate numerical derivatives would be

```
>ffun=function(x) {x^3*sin(x/3)*log(sqrt(x))}
> h=1e-5
>(ffun(x0+h)-ffun(x0))/h
```

What is the approximation error? What will happen if you change h to 1e-10?

```
>
```

Alternatively, we can use a built-in function in *R*.

```
>library(numDeriv)
>options(digits=16)
>grad(ffun, 1, method= “simple” )
>grad(ffun, 1, method= “Richardson” )
```

Which method is better?

The opposite of differentiation is integration. Consider the polynomial function below

$$f(x) = (x-1)^2 + 10x^3 + 5x^4$$

How can we obtain analytical indefinite integration of the polynomial using *R*?

```
> library(PolynomF)
> x=polynom()
> p=(x-1)^2+10*x^3+5*x^4 #What will happen if you skip the previous line?
> integral(p)
```

For numerical results with specified limits

```
> integral(p, limits=c(0, 2))
```

Oftentimes people use *integrate*( ) for integration in *R* is

```
> integrate(p, 0, 2)
```

In-class exercise: A combination of numerical differentiation and integration techniques

$$S_x = 2\pi\int_0^{2\pi} f(x)\sqrt{1+f'(x)^2}\,dx \text{ where } f(x) = \sin(x)$$

What is the value of S_x (the answer you obtain from *R* should be 14.4236)?

The last piece I want address in 9.1 is multiple integrals. Consider the following integral

$$\int_0^1\int_0^1 \frac{1}{1+x^2+y^2}dxdy$$

The first function to tackle this is called *adaptIntegrate*( )

```
> library(cubature)
> f=function(x){1/(1+x[1]^2+x[2]^2)} #The integrand has to be a function of vector
> adaptIntegrate(f, c(0, 0), c(1, 1))
```

One can use the package *pracma* that has two functions – *integral2*( ) and *integral3*( ) – to compute two- and three-dimensional integrals. Unlike *adaptIntegrat*e( ) above, *integral2*( ) needs a function that defines two variables explicitly.

```
> library(pracma)
> f=function(x, y){1/(1+x^2+y^2)}
> integral2(f, 0, 1, 0, 1)
```

Finally, one can solve it as a twofold one-dimensional integral using an intermediate function

```
> fy=function(y){integrate(function(x){1/(1+x^2+y^2)}, 0, 1)$value}
> Fy=Vectorize(fy)
> integrate(Fy, 0, 1)
```

This is much more labor-intensive. It is easier to stick to the early-mentioned methods.

9.2 Numerical optimization

In lecture 5 I have introduced to you the *nlminb*( ) that works very well for MLE. Here I plan to cover other oft-used *R* functions for numerical optimization. If you search on-line, you will probably find a function called *optimize*( ) that is designed for one-dimensional optimization. DO NOT use it and always use *optim*( ) that is developed for multi-dimensional optimization instead. Here is one common test problem for optimization routines in two dimensions – the Rosenbrock "Banana function"

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

When the dimensions are ≤2, it is always a good idea to visualize the function

```
> x1=x2=seq(-1.5,1.5,0.1)
> z=outer(x1, x2, FUN=function(x1, x2){100*(x2-x1^2)^2+(1-x1)^2})
> persp(x1,x2,z,theta=150)
```

Let's define the function and test different optimization routines

```
> fr=function(x){
> x1=x[1]
> x2=x[2]
> 100*(x2-x1^2)^2+(1-x1)^2}
```

We start with the optim( ) and its default Nelder-Mead method

```
> optim(c(0, 0), fr)
```

The Nelder-Mead method is generally robust but slow. An alternative is the BFGS method that will be fast if an analytical form of the gradient of the function is provided. The gradient for the Banana function is

$$\nabla f(x_1, x_2) = \begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{pmatrix}$$

```
> grr=function(x){
> x1=x[1]
> x2=x[2]
> c(-400*x1*(x2-x1^2)-2*(1-x1),200*(x2-x1^2))}
> optim(c(0,0), fr, grr, method="BFGS")
```

The BFGS method finds better answers with fewer trials Another popular routine is the *nlm*( )

```
> nlm(fr, c(0,0))
```

Which method works better?

If we impose two "box" constraints to the Banana function

$$3.5 \le x_1 \le 5 \ \& \ 3.5 \le x_2 \le 5$$

```
> optim(c(4,5), fr, method="L-BFGS-B", lower=c(3.5, 3.5), upper=c(5, 5))
```

How about applying the *nlminb*( )?

```
>
```

Which one is more efficient?

Suppose we have an optimization that involves equality and inequality constraints

$$\begin{aligned} &Min\ \sin(x_1 + x_2 + x_3) \\ &-x_1x_2^3 + x_1^2x_3^2 = 5 \\ &x_1 \ge x_2 \ge x_3. \end{aligned}$$

You need to use the *constrOptim.nl*( ) in the package *alabama*

```
> library(alabama)
> f=function(x){sin(x[1]*x[2]+x[3])}
> heq=function(x){-x[1]*x[2]^3+x[1]^2*x[3]^2-5}
> hin=function(x){ h=c()
> h[1]=x[1]-x[2]; h[2]=x[2]-x[3]
> h}
> ans=constrOptim.nl(c(3,1,0),f, heq, hin)
```

Which will happen if you change the initial guess values?

Let's add a bit complication into the optimization problem

$$\begin{aligned}&Min\ \sin(x_1+x_2+x_3)\\&-x_1x_2^3+x_1^2x_3^2=5\\&x_1\ge x_2\ge x_3\\&5\ge x_1,x_2,x_3\ge 0.\end{aligned}$$

Then you need the *solnp*( ) in the package *Rsolnp*.

```
> library(Rsolnp)
> upper=rep(5, 3)
> lower=rep(0, 3)
> ans=solnp(c(3, 2, 1), f, eqfun=heq, ineqfun=hin, LB=lower, UB=upper,
> ineqLB=c(0, 0), ineqUB=c(5,5))
```

Numerical optimization is fairly straightforward in *R*, isn't it? One more question, how can we obtain answers to *maximization problems* using those functions?

In-class Exercise: Use *constrOptim.nl*( ) and/or *solnp*( ) to solve the optimization problem

$$\begin{aligned}&Min\ f(x_1,x_2)=100(x_2-x_1^2)^2+(1-x_1)^2\\&x_1\le 0.9\\&x_2-x_1\ge 0.1.\end{aligned}$$

```
>
```

9.3 Mathematical programming

Before the end, let's explore how to use *R* to tackle several mathematical programming problems – linear programming (LP), mixed integer linear programming (MILP), and quadratic programming (QP).

The first type LP involves a minimization or maximization problem in which the objective and the constraints can be expressed as **linear** equations or inequalities. In *R*, packages like *lpSolve*, *Rglpk*, *Rsymphony*, and *lpSolveAPI* can handle those problems well. For illustration purposes we use *lpSolve* here. Consider a cost minimization problem

$$\begin{aligned}&Min\ C=5x_1+8x_2\\&x_1+x_2\ge 2\\&x_1+2x_2\ge 3\\&x_1\ge 0,\ x_2\ge 0.\end{aligned}$$

where we assume $x_1, x_2 \in \mathbb{R}^+$ (continuous real numbers), we will relax the assumption soon.

To solve the example problem, you can use *lp*( ) in *lpSolve*.

```
> library(lpSolve)
> simple.lp=lp(objective.in=c(5,8), const.mat=matrix(c(1,1,1,2),2),
const.rhs=c(2,3), const.dir=c(">=",">="))
> simple.lp$solution
```

In-class Exercise: Use *lp*( ) to solve the maximization problem

$$\begin{aligned}&Max\ C = 5x_1 + 8x_2\\&x_1 + x_2 \le 2\\&x_1 + 2x_2 = 3\\&x_1 \ge 0,\ x_2 \ge 0.\end{aligned}$$

```
>
```

Sometimes we have a decision variable *x* that is unrestricted in sign. If that happens, *x* can be written as x_1-x_2 where x_1 & x_2≥0. That is, every unrestricted variable in LP can be replaced by two nonnegative variables. Consider the problem

$$\begin{aligned}&Min\ C = x_1 + 10x_2\\&x_1 + x_2 \ge 2\\&x_1 - x_2 \le 3\\&x_1 \ge 0.\end{aligned}$$

The trick then is to set x_2 as x_3-x_4 and change the problem into

$$\begin{aligned}&Min\ C = x_1 + 10x_3 - 10x_4\\&x_1 + x_3 - x_4 \ge 2\\&x_1 - x_3 + x_4 \le 3\\&x_1 \ge 0, x_3 \ge 0, x_4 \ge 0.\end{aligned}$$

```
>
```

What are the solutions to x_1 & x_2?

In the foregoing examples we assume that all decision variables are continuous. However, in many cases the decision being made must be integer-valued (e.g., people allocated to a store). Then we have a MILP – some decision variables are continuous and some are discrete – or a pure integer programming (IP) problem in which all decision variables must be integers. Consider the following problem:

$$\begin{aligned}&Min\ C = 2x_1 + 3x_2 + 4x_3 - x_4\\&x_1 + 2x_2 \geq 9\\&3x_2 + x_3 \geq 9\\&x_2 + x_4 \leq 10\\&x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0\\&x_2\ \&\ x_4 \in \mathbb{Z}.\end{aligned}$$

```
> integ.LP=lp(objective.in=c(2,3,4,-1), const.mat=matrix(c(1,0,0,2,3,1,
0,1,0,0,0,1),3),const.rhs=c(9,9,10), const.dir=c(">=",">=","<="),                )
```

What is missing above?

The last type is QP, in which the objective function is quadratic and the constraints are linear. That is, we wish to solve the problem

$$\begin{aligned}&Min\, f(x) = -d^T x + \frac{1}{2} x^T Dx\\&A^T \geq x_0\end{aligned}$$

Where x is a column vector, D is a matrix of coefficients, d is a vector of linear coefficients, A is a matrix of constraints, and x_0 is a vector of constraint values. Consider the problem

$$\begin{aligned}&Min\, f(x_1, x_2) = \frac{1}{2} x_1^2 + x_2^2 - x_1 x_2 - 2x_1 - 6x_2\\&x_1 + x_2 \leq 2\\&-x_1 + 2x_2 \leq 2\\&2x_1 + x_2 \leq 3\\&x_1 \geq 0,\ x_2 \geq 0.\end{aligned}$$

You need the *solve.QP*( ) in the package *quadprog*.

```
> library(quadprog)
> Dmat=matrix(c(1,-1,-1, 2), nrow=2) #How do I find the D matrix?
> dvec=c(2,6)
> Amat=matrix(c(-1,-1,1,-2,-2,-1), nrow=2) #How do I get this? Use t( )
> bvec=c(-2,-2,-3)
> solve.QP(Dmat, dvec, Amat, bvec)
```

This example concludes what I aim to teach for numerical methods in *R*. Through this lecture, I hope you will gradually start to appreciate how powerful *R* is when it comes to numerical analysis, in addition to probabilistic and statistical computing.