微積分程式講義

一、 微分函數 D(expr, name)

- Description:
  Compute derivatives of simple expressions, symbolically and algorithmically.

- Arguments:
  (a) expr: an expression or call or (except D) a formula with no lhs.
  (b) name: character vector, giving the variable names (only one for D()) with respect to which derivatives will be computed.

- Example:

```
?D                          #show the usage of function 'D'
a <- expression(x^2+x)      #given an expression
D(a,'x')                    #show derivative of 'x'
x<-2                        #given 'x'
eval(D(a,'x'))              #evaluate the derivative when x=2

f <- D(A*y^3~y) ; f         #y is f's parameter
g <- D(A*y^3~A) ; g         #A is g's parameter
f(y=2,A=1)                  #evaluate the derivative when x=2, A=1
```

```
> ?D
> a <- expression(x^2+x)
> D(a,'x')
2 * x + 1
> x<-2
> eval(D(a,'x'))
[1] 5
>
> f <- D(A*y^3~y) ; f
function (y, A)
A * (3 * y^2)
> g <- D(A*y^3~A) ; g
function (A, y)
y^3
> f(y=2,A=1)
[1] 12
```

二、 積分函數

1、 定積分：Integrate(f, lower, upper…):

- Description:

  Adaptive quadrature of functions of one variable over a finite or infinite interval.

- Arguments:

  (a) f: a R function taking a numeric first argument and returning a numeric vector of the same length. Returning a non-finite element will generate an error.

  (b) lower, upper: the limits of integration. Can be infinite.

- Example:

```
?integrate                                #Integration of One-Dimensional Functions
y <- function(x) {x^2}                    #given function: y=x^2
integrate(y, lower = 0, upper = 10)
integrate(y, 0, 10)                       #preset:2nd is lower, 3rd is uppeer
```

```
> ?integrate
> y <- function(x) {x^2}
> integrate(y, lower = 0, upper = 10)
333.3333 with absolute error < 3.7e-12
> integrate(y, 0, 10)
333.3333 with absolute error < 3.7e-12
```

2、 不定積分：antiD()

- Description:

  Operators for computing anti-derivatives as functions.

- Example:

```
library(mosaicCalc)          #'antiD' is in mosaicCalc package
?antiD                       #Anti-derivative
antiD( a*x^2 ~ x)            #a is constant
```

```
> ?antiD
> antiD( a*x^2 ~ x)
function (x, C = 0, a)
a * 1/3 * x^3 + C
```

三、 其他函數

1、 指對數：

- Description:

  (a) log computes logarithms, by default natural logarithms, log10 computes common (i.e., base 10) logarithms, and log2 computes binary (i.e., base 2) logarithms. The general form log(x, base) computes logarithms with base base.
  log1p(x) computes log(1+x) accurately also for |x| << 1.

  (b) exp computes the exponential function.
  expm1(x) computes exp(x) - 1 accurately also for |x| << 1.

- Example:

```
?log                    #preset:base=exp(1)
log(exp(1))

log10(100)              #log_():base=_     ] same
log(100,base=10)        #log(,base=)
```

```
> ?log
> log(exp(1))
[1] 1
>
> log10(100)
[1] 2
> log(100,base=10)
[1] 2
```

2、 絕對值、根號：

- Description:
  abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x, √{x}.

- Example:

```
x=-4
abs(x)
sqrt(abs(x))   #we can't use negative x in sqrt
```

```
> x=-4
> abs(x)
[1] 4
> sqrt(abs(x))
[1] 2
```

3、 三角、反三角：

- Description:
  These functions give the obvious trigonometric functions. They respectively compute the cosine, sine, tangent, arc-cosine, arc-sine, arc-tangent, and the two-argument arc-tangent.
  cospi(x), sinpi(x), and tanpi(x), compute cos(pi*x), sin(pi*x), and tan(pi*x).

- Example:

```
?pi            #math.pi
?tanpi         #tan(pi*x)
sin(pi/2)
asin(1)        #1.570796=pi/2
cos(0)
tan(pi/4)
cospi(1)
```

```
> ?pi
> ?tanpi
> sin(pi/2)
[1] 1
> asin(1)
[1] 1.570796
> cos(0)
[1] 1
> tan(pi/4)
[1] 1
> cospi(1)
[1] -1
```

四、 繪圖函數

1、 makeFun:

- Description:
  Provides an easy mechanism for creating simple "mathematical" functions via a formula interface.

- Example:

```
library(mosaic)                #'makeFun' is in mosaic package
?makeFun                       #Create a function from a formula

a <- makeFun(x^2+x~x);a
f <- makeFun( sin(x^2 * b) ~ x & y & a); f
g <- makeFun( sin(x^2 * b) ~ x & y & a, a=2 ); g
h <- makeFun( a * sin(x^2 * b) ~ b & y, a=2, y=3); h
```

```
> library(mosaic)
> ?makeFun
>
> a <- makeFun(x^2+x~x);a
function (x)
x^2 + x
> f <- makeFun( sin(x^2 * b) ~ x & y & a); f
function (x, y, a, b)
sin(x^2 * b)
> g <- makeFun( sin(x^2 * b) ~ x & y & a, a=2 ); g
function (x, y, a = 2, b)
sin(x^2 * b)
> h <- makeFun( a * sin(x^2 * b) ~ b & y, a=2, y=3); h
function (b, a = 2, y = 3, x)
a * sin(x^2 * b)
```

2、 plotFun:

- Description:

  Plots mathematical expressions in one and two variables.

- Example:

```
?plotFun                                             #Plotting mathematical expressions
a <- makeFun(x^2+x~x)
plotFun( a, xlim = range(0,10))                      #use the function we made from 'makeFun'
plotFun( a*sin(x^2)~x, xlim=range(-5,5), a=2 )       #setting parameter value
plotFun( u^2 ~ u, ulim=c(-4,4) )                     #ulim here = x-axis = xlim
plotFun( y^2 ~ y, ylim=c(-2,20), y.lim=c(-4,4) ) #ylim:y-axis ; y.lim:parameter(x-axis)
```