Option Pricing Functions to Accompany *Derivatives Markets*

Robert McDonald

May 8, 2016

Contents

1 Introduction

This vignette is an overview to the functions in the functions in the *derivmkts* package, which was conceived as a companion to my book *Derivatives Markets*. The material has an educational focus. There are other option pricing packages for R, but this package has several distinguishing features:

- function names (mostly) correspond to those in *Derivatives Markets*.

Function	Description
bscall	European call
bsput	European put
bsopt	European call and put and associated Greeks: delta, gamma, vega, theta, rho, psi, and elasticity
assetcall	Asset-or-nothing call
assetput	Asset-or-nothing put
cashcall	Cash-or-nothing call
cashput	Cash-or-nothing put

Table 1: Black-Scholes related option pricing functions

- vectorized Greek calculations are convenient both for individual options and for portfolios
- the `quincunx` function illustrates the workings of a quincunx (Galton board).
- binomial functions include a plotting function that provides a visual depiction of early exercise

2 European Calls and Puts

Table 1 lists the Black-Scholes related functions in the package. The functions `bscall`, `bsput`, and `bsopt` provide basic pricing of European calls and puts. There are also options with binary payoffs: cash-or-nothing and asset-or-nothing options. All of these functions are vectorized. The function `bsopt` by default provides option greeks. Here are some examples:

```
s <- 100; k <- 100; r <- 0.08; v <- 0.30; tt <- 2; d <- 0
bscall(s, k, v, r, tt, d)

[1] 24.02

bsput(s, c(95, 100, 105), v, r, tt, d)

[1]  7.488  9.239 11.188
```

3 Barrier Options

There are pricing functions for the following barrier options:

- down-and-in and down-and-out barrier binary options
- up-and-in and up-and-out barrier binary options
- more standard down- and up- calls and puts, constructed using the barrier binary options

Naming for the barrier options generally follows the convention

`[u|d][i|o][call|put]`

which means that the option is "up" or "down", "in" or "out", and a call or put.[1] An up-and-in call, for example, would be denoted by `uicall`. For binary options, we add the underlying, which is either the asset or $1: cash:

`[asset|cash][u|d][i|o][call|put]`

```
H <- 115
bscall(s, c(80, 100, 120), v, r, tt, d)

[1] 35.28 24.02 15.88

uicall(s, c(80, 100, 120), v, r, tt, d, H)

[1] 34.55 23.97 15.88

bsput(s, c(80, 100, 120), v, r, tt, d)

[1]  3.450  9.239 18.141

uoput(s, c(80, 100, 120), v, r, tt, d, H)

[1] 2.328 5.390 9.070
```

4 Option Greeks

Greeks for vanilla and barrier options can be computed using the `greeks` function, which is a wrapper for any pricing function that returns the option price and which uses the default naming of inputs.[2]

```
H <- 105
greeks(uicall(s, k, v, r, tt, d, H))

              uicall
Price      24.023834
Delta       0.722328
Gamma       0.007903
Vega        0.474353
Rho         0.963838
Theta      -0.020310
Psi        -1.444314
Elasticity  3.006716
```

[1] This naming convention differs from that in *Derivatives Markets*, in which names are `callupin`, `callupout`, etc. Thus, I have made both names are available for these functions.

[2] In this version of the package, I have two alternative functions that return Greeks:

- The `bsopt` function by default produces prices and Greeks for European calls and puts.
- The `greeks2` function takes as arguments the name of the pricing function and then inputs as a list.

These may be deprecated in the future.

The value of this approach is that you can easily compute Greeks for spreads and custom pricing functions. Here are two examples Here is the formula for a prepaid contract that pays S_T^a at time T:

```
powercontract <- function(s, v, r, tt, d, a) {
    price <- exp(-r*tt)*s^a* exp((a*(r-d) + 1/2*a*(a-1)*v^2)*tt)
}
greeks(powercontract(s=40, v=.08, r=0.08, tt=0.25, d=0, a=2))

           powercontract
Price           1634.936
Delta             81.747
Gamma              2.044
Vega               0.654
Rho                4.087
Theta             -0.387
Psi               -8.175
Elasticity         2.000
```

Compute the greeks for a spread by defining the value of the spread as a function, and then computing the greeks for the function:

```
bullspread <- function(s, v, r, tt, d, k1, k2) {
    bscall(s, k1, v, r, tt, d) - bscall(s, k2, v, r, tt, d)
}
greeks(bullspread(39:41, .3, .08, 1, 0, k1=40, k2=45))

           bullspread_39 bullspread_40 bullspread_41
Price          2.0020318     2.1551927     2.306e+00
Delta          0.1542148     0.1519426     1.487e-01
Gamma         -0.0017692    -0.0027545    -3.614e-03
Vega          -0.0080732    -0.0132218    -1.822e-02
Rho            0.0401235     0.0392251     3.793e-02
Theta         -0.0005476    -0.0003164    -8.246e-05
Psi           -0.0601438    -0.0607771    -6.099e-02
Elasticity     3.0041376     2.8200287     2.645e+00
```

The Greeks function is vectorized, so you can create vectors of greek values with a single call. This example plots, for a bull spread, the gamma as a function of the stock price.

```
sseq <- seq(1, 100, by=0.5)
x <- greeks(bullspread(sseq, .3, .08, 1, 0, k1=40, k2=45))
plot(sseq, x['Gamma',], type='l')
```

Here is a final example:

```
k <- 100; r <- 0.08; v <- 0.30; tt <- 2; d <- 0
S <- seq(.5, 250, by=.5)
cgreeks <- greeks(bscall(S, k, v, r, tt, d))
pgreeks <- greeks(bsput(S, k, v, r, tt, d))
optlbl <-  c('Call', 'Put')
y <- list(cgreeks, pgreeks)
par(mfrow=c(4, 4))  ## create a 4x4 plot
par(mar=c(2,2,2,2))
for (i in 1:length(y)) {
```

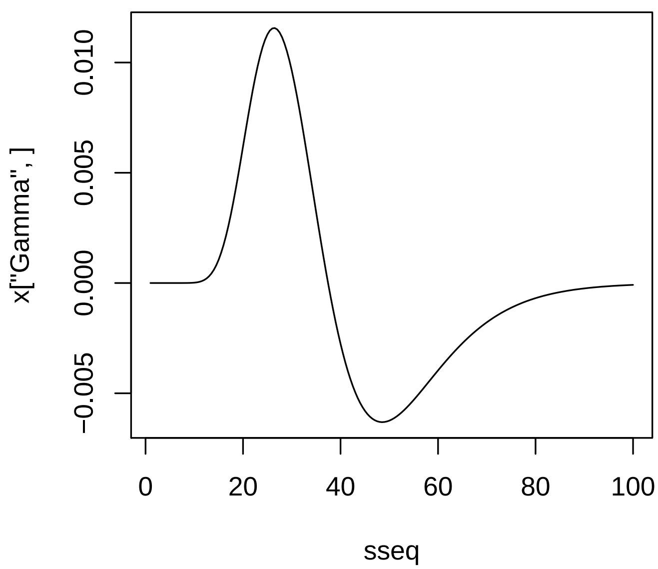

Figure 1: Gamma for a 40-45 bull spread.

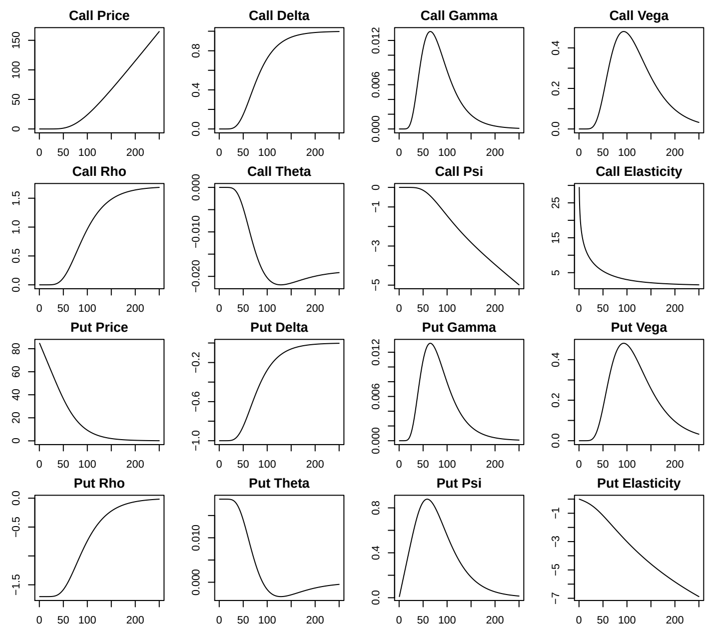

Figure 2: All option Greeks, plotted using bsopt

```
    for (j in rownames(y[[i]])) {  ## loop over greeks
        plot(S, y[[i]][j, ], main=paste(optlbl[i], j), ylab=j, type='l')
    }
}
```

5 Binomial Pricing of European and American Options

There are two functions related to binomial pricing:

binomopt computes prices of American and European calls and puts. The function has three optional parameters that control output:

- `returnparams=TRUE` will return as a vector the option pricing inputs, computed parameters, and risk-neutral probability.
- `returngreeks=TRUE` will return as a vector the price, delta, gamma, and theta at the initial node.
- `returntrees=TRUE` will return as a list the price, greeks, the full stock price tree, the exercise status (`TRUE` or `FALSE`) at each node, and the replicating portfolio at each node.

binomplot displays the asset price tree, the corresponding probability of being at each node, and whether or not the option is in exercised at each node. This function is described in more detail in Section 9.2.

Here are examples of pricing, illustrating the default of just returning the price, and the ability to return the price plus parameters, as well as the price, the parameters, and various trees:

```
s <- 100; k <- 100; r <- 0.08; v <- 0.30; tt <- 2; d <- 0.03
binomopt(s, k, v, r, tt, d, nstep=3)

price
 20.8

binomopt(s, k, v, r, tt, d, nstep=3, returnparams=TRUE)

   price        s        k        v        r       tt        d    nstep
 20.7961 100.0000 100.0000   0.3000   0.0800   2.0000   0.0300   3.0000
       p       up       dn        h
  0.4391   1.3209   0.8093   0.6667

binomopt(s, k, v, r, tt, d, nstep=3, putopt=TRUE)

price
12.94

binomopt(s, k, v, r, tt, d, nstep=3, returntrees=TRUE, putopt=TRUE)

$price
price
12.94

$greeks
    delta     gamma     theta
-0.335722  0.010614 -0.007599

$params
       s        k        v        r       tt        d    nstep        p
100.0000 100.0000   0.3000   0.0800   2.0000   0.0300   3.0000   0.4391
      up       dn        h
  1.3209   0.8093   0.6667

$oppricetree
      [,1]   [,2]   [,3]  [,4]
[1,] 12.94  3.816  0.000  0.00
[2,]  0.00 21.338  7.176  0.00
[3,]  0.00  0.000 34.507 13.49
[4,]  0.00  0.000  0.000 47.00

$stree
     [,1]   [,2]   [,3]   [,4]
[1,]  100 132.09 174.47 230.45
[2,]    0  80.93 106.89 141.19
[3,]    0   0.00  65.49  86.51
[4,]    0   0.00   0.00  53.00
```

```
$probtree
     [,1]   [,2]   [,3]    [,4]
[1,]    1 0.4391 0.1928 0.08464
[2,]    0 0.5609 0.4926 0.32441
[3,]    0 0.0000 0.3146 0.41445
[4,]    0 0.0000 0.0000 0.17650

$exertree
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE
[3,] FALSE FALSE  TRUE  TRUE
[4,] FALSE FALSE FALSE  TRUE

$deltatree
        [,1]    [,2]    [,3]
[1,] -0.3357 -0.1041  0.0000
[2,]  0.0000 -0.6471 -0.2419
[3,]  0.0000  0.0000 -0.9802

$bondtree
      [,1]  [,2]  [,3]
[1,] 46.51 17.56  0.00
[2,]  0.00 73.71 33.03
[3,]  0.00  0.00 94.81
```

6 Asian Options

There are analytical functions for valuing geometric Asian options and Monte Carlo routines for valuing arithmetic Asian options.

6.1 Geometric Asian Options

Geometric Asian options can be valued using the Black-Scholes formulas for vanilla calls and puts, with modified inputs. The functions return both call and put prices with a named vector:

```
geomavgprice(s, k, v, r, tt, d, 3)

  Call    Put
13.123  8.455

geomavgstrike(s, k, v, r, tt, d, 3)

 Call   Put
9.058 4.764
```

6.2 Arithmetic Asian Options

Monte Carlo valuation is used to price arithmetic Asian options. For efficiency, the function `arithasianmc` returns call and put prices for average price and average strike options. By default the number of simulations is 1000. Optionally the function returns the standard deviation of each estimate

```
arithasianmc(s, k, v, r, tt, d, 3, numsim=5000, printsds=TRUE)

              Call    Put sd Call sd Put
Avg Price  13.870  8.054   21.88 11.413
Avg Strike  8.212  5.048   13.92  7.426
Vanilla    19.771 10.791   32.68 14.869
```

The function `arithavgpricecv` uses the control variate method to reduce the variance in the simulation. At the moment this function prices only calls, and returns both the price and the regression coefficient used in the control variate correction:

```
arithavgpricecv(s, k, v, r, tt, d, 3, numsim=5000)

 price   beta
13.949  1.041
```

7 Jumps and Stochastic Volatility

The `mertonjump` function returns call and put prices for a stock that can jump discretely. A poisson process controls the occurrence of a jump and the size of the jump is lognormally distributed. The parameter `lambda` is the mean number of jumps per year, the parameter `alphaj` is the log of the expected jump, and `sigmaj` is the standard deviation of the log of the jump. The jump amount is thus drawn from the distribution

$$Y \sim \mathcal{N}(\alpha_J - 0.5\sigma_J^2, \sigma_J^2)$$

```
mertonjump(s, k, v, r, tt, d, lambda=0.5, alphaj=-0.2, vj=0.3)

 Call   Put
23.98 15.02

c(bscall(s, k, v, r, tt, d), bsput(s, k, v, r, tt, d))

[1] 19.96 11.00
```

8 Bonds

The simple bond functions provided in this version compute the present value of cash flows (`bondpv`), the IRR of the bond (`bondyield`), Macaulay duration (`duration`), and convexity (`convexity`).

```
coupon <- 8; mat <- 20; yield <- 0.06; principal <- 100;
modified <- FALSE; freq <- 2
price <- bondpv(coupon, mat, yield, principal, freq)
price

[1] 123.1

bondyield(price, coupon, mat, principal, freq)

[1] 0.06

duration(price, coupon, mat, principal, freq, modified)

[1] 11.23

convexity(price, coupon, mat, principal, freq)

[1] 170.3
```

9 Functions with Graphical Output

Several functions provide visual illustrations of some aspects of the material.

9.1 Quincunx or Galton Board

The quincunx is a physical device the illustrates the central limit theorem. A ball rolls down a pegboard and strikes a peg, falling randomly either to the left or right. As it continues down the board it continues to strike a series of pegs, randomly falling left or right at each. The balls collect in bins and create an approximate normal distribution.

The quincunx function allows the user to simulate a quincunx, observing the path of each ball and watching the height of each bin as the balls accumulate. More interestingly, the quincunx function permits altering the probability that the ball will fall to the right.

Figure 3 illustrates the function after dropping 200 balls down 20 levels of pegs with a 70% probability that each ball will fall right:

```
par(mar=c(2,2,2,2))
quincunx(n=20, numballs=200, delay=0, probright=0.7)
```

9.2 Plotting the Solution to the Binomial Pricing Model

The `binomplot` function calls `binomopt` to compute the option price and the various trees, which it then uses in plotting:

The first plot, figure 4, is basic:

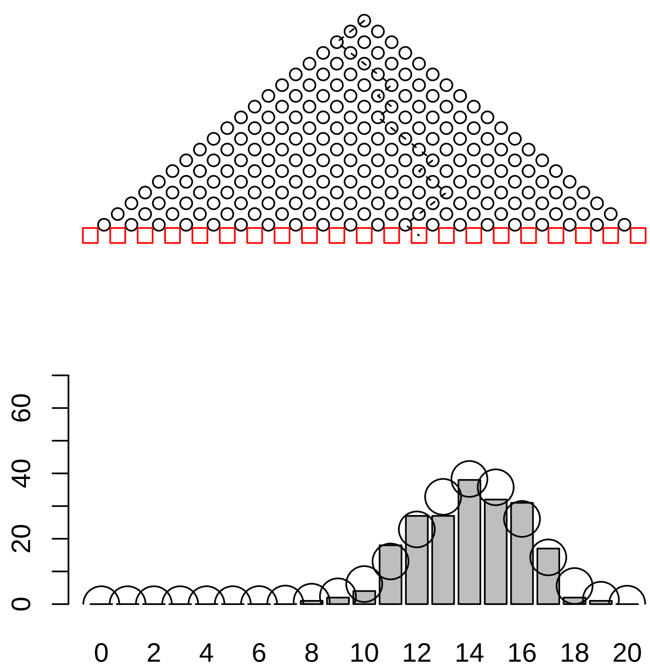

Figure 3: Output from the Quincunx function

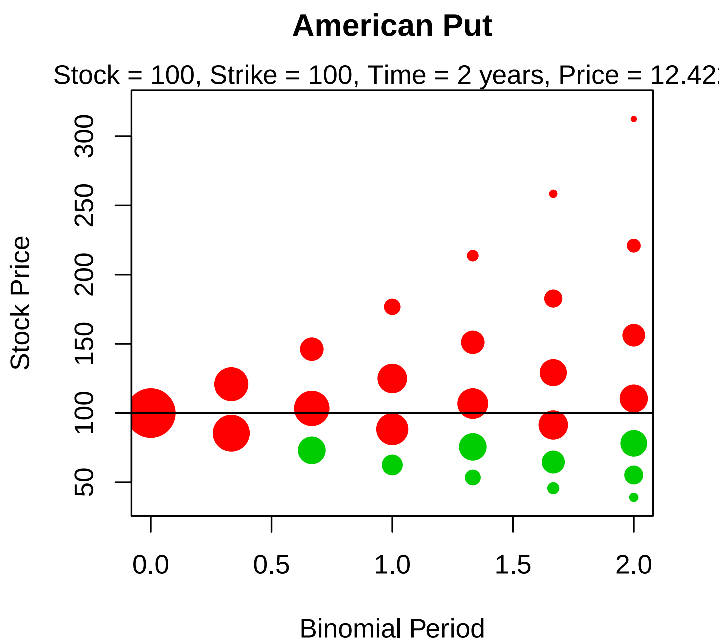

Figure 4: Basic option plot showing stock prices and nodes at which the option is exercised.

```
binomplot(s, k, v, r, tt, d, nstep=6, american=TRUE, putopt=TRUE)
```

The second plot, figure 5, adds a display of stock prices and arrows connecting the nodes.

```
binomplot(s, k, v, r, tt, d, nstep=6, american=TRUE, putopt=TRUE,
    plotvalues=TRUE, plotarrows=TRUE)
```

As a final example, consider an American call when the dividend yield is positive and `nstep` has a larger value. Figure 6 shows the plot, with early exercise evident.

```
d <- 0.06
binomplot(s, k, v, r, tt, d, nstep=40, american=TRUE)
```

The large value of `nstep` creates a high maximum terminal stock price, which makes details hard to discern in the boundary region where exercise first occurs. We can zoom in on that region by selecting values for `ylimval`; the result is in Figure 7.

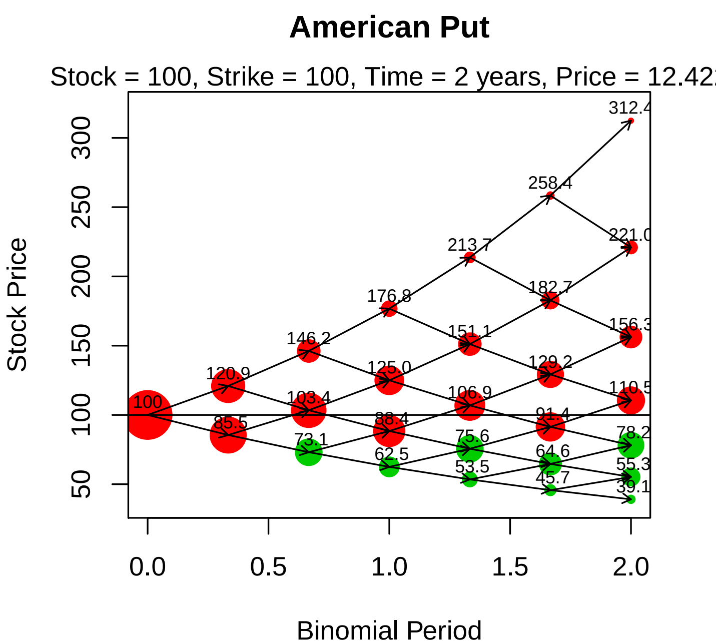

Figure 5: Same plot as Figure 4 except that values and arrows are added to the plot.

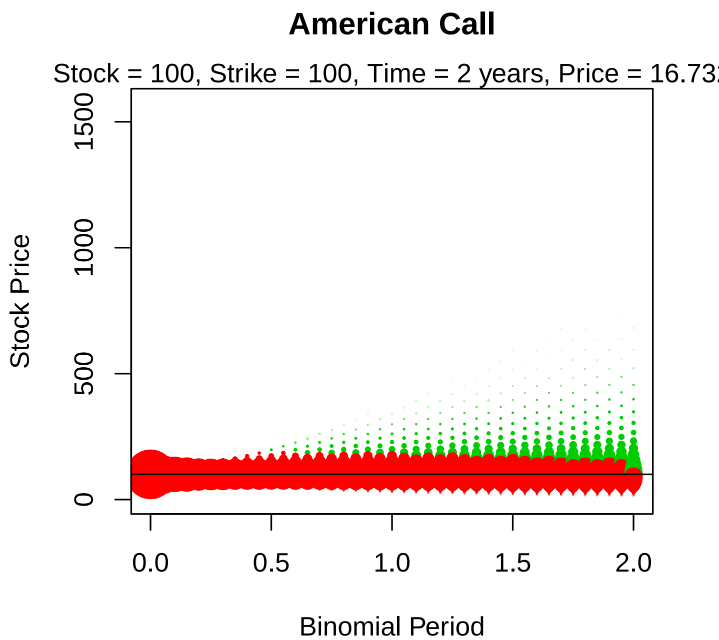

Figure 6: Binomial plot when nstep is 40.

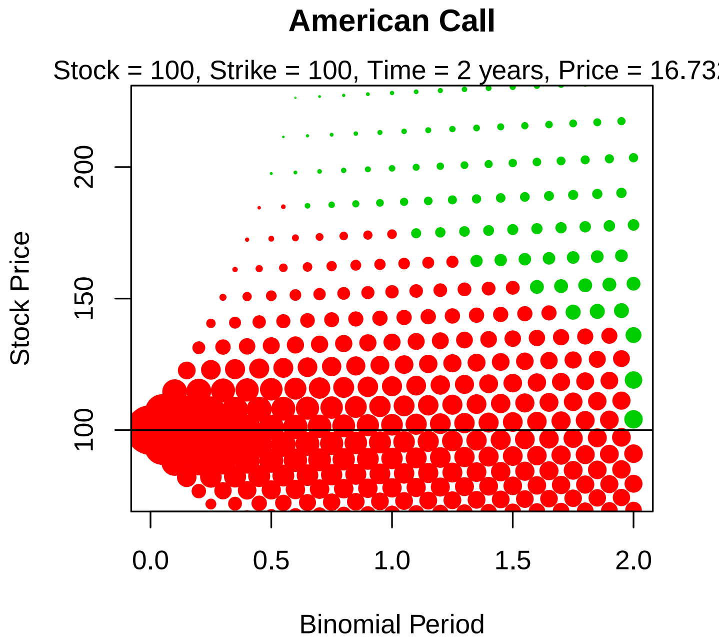

Figure 7: Binomial plot when nstep is 40 using the argument ylimval to focus on a subset.

```
d <- 0.06
binomplot(s, k, v, r, tt, d, nstep=40, american=TRUE, ylimval=c(75, 225))
```