Distance to default package

Benjamin Christoffersen

April 27, 2018

This package is provides fast functions to work with the Merton's distance to default model. We will denote the observed market values by S_t and unobserved asset values by V_t. We assume that V_t follows a geometric Brownian motion

$$dV_t = \mu V_t \,\mathrm{d}t + \sigma V_t \,\mathrm{d}W_t$$

We assume that we observe the assets over increaments of dt in time. Thus, we will let V_k and V_{k+1} be the value at $t_0 + k \cdot dt$. Thus,

$$V_{k+1} = V_k \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) dt + \sigma W_t\right)$$

We further let r denote the risk free rate, D_t denote debt due at time $t + T$. Then

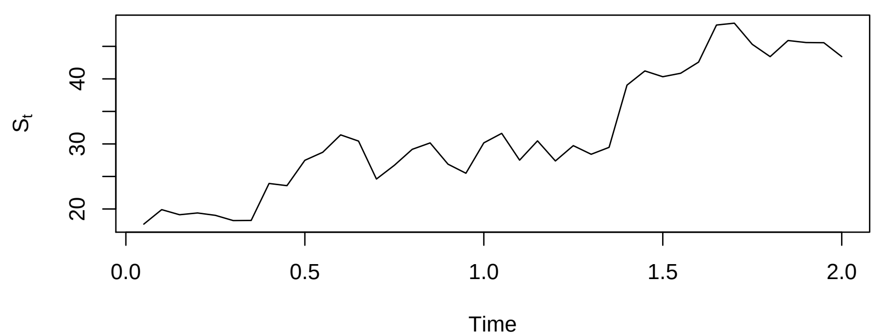
$$\begin{aligned}
C(V_t, D_t, T, \sigma, r) &= V_t N(d_1) - D_t \exp(-rT)\, N(d_1 - \sigma\sqrt{T}) \\
d_1 &= \frac{\log(V_t) - \log D_t + \left(r + \frac{1}{2}\sigma^2\right) T}{\sigma\sqrt{T}} && (1) \\
S_t &= C(V_t, D_t, T, \sigma, r) && (2)
\end{aligned}$$

where C is a European call option. d_1 in equation (1) is the so-called distance to default. Equation (2) can be computed with the `BS_call` function. Further, the `get_underlying` can be used to invert the equation (2)

```r
library(DtD)
(S <- BS_call(100, 90, 1, .1, .3))

## [1] 22.51

get_underlying(S, 90, 1, .1, .3)

## [1] 100
```

To illustrate the above then we can simulate the underlying and transform the data into the stock price as follows

```r
# assign parameters
std <- .1
mu  <- .05
dt  <- .05
V_0 <- 100
t.  <- seq(dt, 2, by = dt)
D   <- c(rep(80, 27), rep( 70, length(t.) - 27))
r   <- c(rep( 0, 13), rep(.02, length(t.) - 13))

# simulate underlying
set.seed(83992673)
V <- V_0 * exp(
  (mu - std^2/2) * t. + cumsum(rnorm(length(t.), sd = std * sqrt(dt))))

# compute stock price
S <- mapply(BS_call, V, D, T = 1, r, std)
plot(t., S, type = "l", xlab = "Time", ylab = expression(S[t]))
```



Despite that the model assume a constant risk free rate than we let it vary in this example. We end by plotting the stock price. Further, we can confirm that we the same underlying after transforming back

```r
all.equal(V, get_underlying(S, D, 1, r, std))

## [1] TRUE
```

1 Drift and volatility estimation