Black-Derman-Toy Interest Rate Tree (R)

```
# bdt.R
require(combinat)

# calibration data
yields = c(0.10, 0.11, 0.12, 0.125)
volatilities = c(NA, 0.10, 0.15, 0.14)

bdt.tree <- function(yields, volatilties) {
  prices <- numeric(0)
  for(i in 1:length(yields)) prices = c(prices, (1+yields[i])^(-1*i))

  # method for determining roots of n equations with n unknowns
  # approximate partial derivatives using defn of derivative
  multiNewtons <- function(fn, x, iterations=10) {
    f <- function(x) {
      values <- numeric(0)
      for(i in 1:length(fn)) {
        values = c(values, fn[[i]](x))
      }
      t(t(values))
    }
    J <- function(x) {
      h = 0.00001 # decreasing h increasing accuracy of approx partial derivative
      partials <- numeric(0)
      for(i in 1:length(fn)) { # row
        for(j in 1:length(fn)) { # column
          ej <- numeric(0) # unit vector
          for(k in 1:length(fn)) {
            if(k==j) ej = c(ej, 1)
            else ej = c(ej, 0)
          }
          partials = c(partials, (fn[[i]](x + h*ej) - fn[[i]](x))/h) # approx partial
              derivative
        }
      }
      matrix(partials, nrow=i, ncol=j, byrow=TRUE)
    }

    temp <<- fn
    for(i in 1:iterations) {
      x = x - solve(J(x))%*%f(x)
    }
    x
  }

  P <- function(i=1, j, node, params, rateTree) {
    R = params[1]
    sigma = params[2]

    if(j==2) {
      if(node==1) {
```

```
    (1+R*exp(2*sigma))^-1
  } else {
    (1+R)^-1
  }
} else { # j = 3, 4, ...
  pathCounter = 1
  paths = list()

  for(m in 0:(j-2)) {
    path = c(rep(0, m), rep(1, (j-2)-m))
    permPaths = unique(permn(path))
    for(n in 1:length(permPaths)) {
      paths[[pathCounter]] = permPaths[[n]]
      pathCounter = pathCounter + 1
    }
  }

  total = 0
  prob = 0.5^(j-2)
  for(m in 1:length(paths)) {
    path = paths[[m]]

    levelIndex = length(path)+2 # year - 1
    upIndex = 1 # 1 implies no down movements
    if(node==0) {
      upIndex = upIndex + 1
    }

    for(n in 1:length(path)) {
      if(path[n] == 0) {
        upIndex = upIndex + 1
      }
    }

    dFactors <- numeric(0) # use the path to find the discount factors
    for(n in length(path):1) { # iterate backwards from
        rateTree[[upIndex]][levelIndex] through the path
      levelIndex = levelIndex - 1
      if(path[n] == 0) { # a movement down implies a movement up when going backwards
        upIndex = upIndex - 1
      } # else don't change upIndex
      dFactors = c(dFactors, rateTree[[levelIndex]][upIndex])
    }

    # apply dFactors to rateTree[[upIndex]][levelIndex]
    product = 1
    for(l in 1:length(dFactors)) {
      product = product * (1+dFactors[l])^-1
    }
    tree <<- product # comment this out
    total = total + product * (prob*(1 + R*exp(2*sigma*(sum(path)+node)))^-1)
  }
  total
}
```

```
  }

  # determine the BDT short rate tree
  rateTree = list()
  for(i in 1:length(yields)) {
    if(i == 1) {
      rateTree[[i]] = yields[i]
    } else {
      params = c(yields[i], volatilities[i]) # initial estimate

      f1 <- function(params) {
        (1 + yields[1])^-1 * (1/2) * (P(1,i,1,params,rateTree) +
            P(1,i,0,params,rateTree)) - prices[i]
      }
      f2 <- function(params) {
        (1/2) * log( (P(1,i,1,params,rateTree)^(-1/(i-1)) -
            1)/(P(1,i,0,params,rateTree)^(-1/(i-1)) - 1) ) - volatilities[i]
      }
      fn = c(f1, f2)

      params = multiNewtons(fn, params) # better estimate
      R = params[1]
      sigma = params[2]

      rates <- numeric(0)
      for(j in i:1) {
        rates = c(rates, R*exp(2*(j-1)*sigma))
      }
      rateTree[[i]] = rates
    }
  }

  rateTree
}

rateTree = bdt.tree(yields, volatilities)

# Output
> rateTree
[[1]]
[1] 0.1

[[2]]
[1] 0.1322011 0.1082371

[[3]]
[1] 0.20170244 0.13662290 0.09254136

[[4]]
[1] 0.20028379 0.15683226 0.12280753 0.09616446
```