Finance Quantitative

TP 1: Courbe zéro-coupon et modèle de Black-Derman-Toy

Patrick Hénaff

Version: 22 Mar 2025

Contents

Ce projet comporte trois parties: - Calcul d'une courbe zéro-coupon à partir d'un échantillon d'obligations - Construction d'un modèle de Black-Derman-Toy - Calcul du prix de quelques obligations comportant des clauses optionnelles

1 Calcul d'une courbe zéro-coupon

On se propose d'estimer une courbe de taux zéro-coupon en utilisant le modèle de [@Swensson1994]. Ce modèle est utilisé par la Banque Centrale Européenne pour publier quotidiennement la courbe de taux des emprunts d'Etat de la zone Euro.

Le taux zéro-coupon de maturité T est donné par l'expression:

Dt Maturité	Coupon (%)	Prix	Id
2025-07-14	4.6	100.894	Bond-1
2025-08-05	1.5	99.616	Bond-2
2026-09-02	3.7	102.400	Bond-3
2026-04-27	3.4	101.399	Bond-4
2027-03-22	1.4	98.802	Bond-5
2027-09-07	4.8	106.609	Bond-6
2028-03-14	5	108.310	Bond-7
2028-12-27	2.6	101.826	Bond-8
2030-03-19	1.7	97.247	Bond-9
2031-05-05	3.9	109.524	Bond-10

$$z(t) = \beta_0 + \beta_1 \frac{1 - \exp -t/\tau_1}{t/\tau_1} \tag{1}$$

$$+ \beta_2 \left(\frac{1 - \exp -t/\tau_1}{t/\tau_1} - \exp -t/\tau_1 \right) \tag{2}$$

$$+ \beta_3 \left(\frac{1 - \exp -t/\tau_2}{t/\tau_2} - \exp -t/\tau_2 \right) \tag{3}$$

Les données relatives aux obligations sont disponibles dans le fichier “prix_obligations.csv”. Un extrait figure dans le tableau ci-dessous. Pour chaque titre on dispose de la date de maturité, du coupon en %, et du prix “pied de coupon” observé le 20 février 2025, pour 100 de nominal.

Les étapes du calcul:

- Pour chaque titre, calculer le prix “coupon plein”, c’est à dire la valeur actualisée des flux futurs. On rappelle que le prix “coupon plein” est le égal au prix “pied de coupon” plus le coupon couru. Utiliser la convention ACT/ACT pour le décompte des jours.
- Ecrire une fonction qui calcule l’échéancier des paiements futurs de chaque obligation. Pour déterminer la date de chaque paiement, on utilisera la règle “Modified Following”, en ne prenant en compte que les fins de semaines. On ne prend pas en compte des jours fériés.
- Ecrire une fonction qui calcule la valeur actualisée de chaque obligation en fonction des paramètres du modèle de Swensson. Le modèle de Swensson donne un taux continu.
- Finalement, déterminer les 6 paramètres du modèle par minimisation de l’erreur globale de valorisation. Comme les obligations de l’échantillon ont des maturité très différentes, on pourra normaliser l’erreur sur chaque titre par la duration du titre.

Pour présenter les résultats, on pourra reporter sur un graphe les prix (ou les rendements) calculés et les prix (ou les rendements) observés des obligations.

Solution

La règle “modified following” est appliquée pour s’assurer que les dates de paiement tombent un jour ouvré:

- Si la date de paiement tombe un jour non-ouvré, avancer au prochain jour ouvré
- Si la règle précédente fait changer de mois, reculer au jour oublié précédent la date de paiement.

```r
# adjust to next business days - modified following rule ignoring holidays
adjust_mf <- function(date) {
    adjusted.date <- date
    if (!is.bizday(adjusted.date, cal="weekends")) {
      adjusted.date <- add.bizdays(adjusted.date, 1, cal="weekends")
      if (month(adjusted.date) != month(date)) {
        # modified following rule: do not move to next month
        adjusted.date <- add.bizdays(adjusted.date, -1, cal="weekends")
      }
    }
    adjusted.date
}
```

On utilise cette fonction pour calculer l'échéancier d'une obligation:

```r
generate_bond_cashflows <- function(pricing_date, maturity_date, coupon_rate) {
  # Input validation
  if (!is.Date(pricing_date) || !is.Date(maturity_date)) {
    stop("Pricing date and maturity date must be Date objects.")
  }
  if (maturity_date <= pricing_date) {
    stop("Maturity date must be after the pricing date.")
  }
  if (!is.numeric(coupon_rate) || coupon_rate < 0) {
    stop("Coupon rate must be a non-negative number.")
  }

  # Extract day and month of maturity date
  maturity_day <- day(maturity_date)
  maturity_month <- month(maturity_date)

  # previous coupon date, used for computing accrued interest
  dt <- ymd(paste(year(pricing_date),maturity_month, maturity_day, sep="-"))
  dt <- adjust_mf(dt)
  if(dt >= pricing_date) {
    dt <- ymd(paste(year(pricing_date)-1,maturity_month, maturity_day, sep="-"))
    dt <- adjust_mf(dt)
  }
  previous_coupon_date <- dt

  # Generate coupon years
  coupon_years <- seq(year(pricing_date), year(maturity_date), by = 1)

  # Create coupon dates using the extracted day and month
  coupon_dates <- ymd(paste(coupon_years, maturity_month, maturity_day, sep = "-"))

  # Remove coupon dates before the pricing date
  coupon_dates <- coupon_dates[coupon_dates >= pricing_date]

  # Adjust for week-ends ("modified following") - Ignore holidays for simplicity
  adjusted_coupon_dates <- sapply(coupon_dates, adjust_mf)
  adjusted_coupon_dates <- as.Date(adjusted_coupon_dates, origin = "1970-01-01")
```

```
  cash_flow_amounts <- rep(coupon_rate, length(adjusted_coupon_dates))
  cash_flow_amounts[length(cash_flow_amounts)] <- cash_flow_amounts[length(cash_flow_amounts)] + 100

  cash_flows_df <- data.frame(
    date = adjusted_coupon_dates,
    amount = cash_flow_amounts
  )
  list(cf=cash_flows_df, dt_previous_coupon=previous_coupon_date)
}
```

La fonction suivante calcule le taux zéro-coupon, fonction de la maturité.

Pour calculer la valeur actualisée d'une obligation, on actualise chaque flux au taux zéro-coupon correspondant:

```
dt.pricing <- ymd(20250217)

bond.calc <- function(params, cashflows, calc.duration=FALSE) {
  # maturity in fraction of year
  ttm <- as.numeric(difftime(cashflows$date, dt.pricing)) / 365.25
  zc <- zc.Swensson(params, ttm)/100
  discount.factor <- exp(-zc*ttm)
  pv <- discount.factor * cashflows$amount
  res <- list(pv=sum(pv))
  if(calc.duration) {
    res$duration <- sum(ttm*pv) / res$pv
  }
  res
}
```

Dans la fonction objectif, on normalise l'écart de prix par la duration.

```
loss.function <- function(p, portfolio) {
  loss <- 0
  params <- list(beta0=p[1], beta1=p[2], beta2=p[3],
                 beta3=p[4], tau1=p[5], tau2=p[6])
  for(bond.data in portfolio) {
    res <- bond.calc(params, bond.data$cf, calc.duration = TRUE)
    er <- (res$pv - bond.data$pv) / res$duration
    loss <- loss + er^2
  }
  loss
}
```

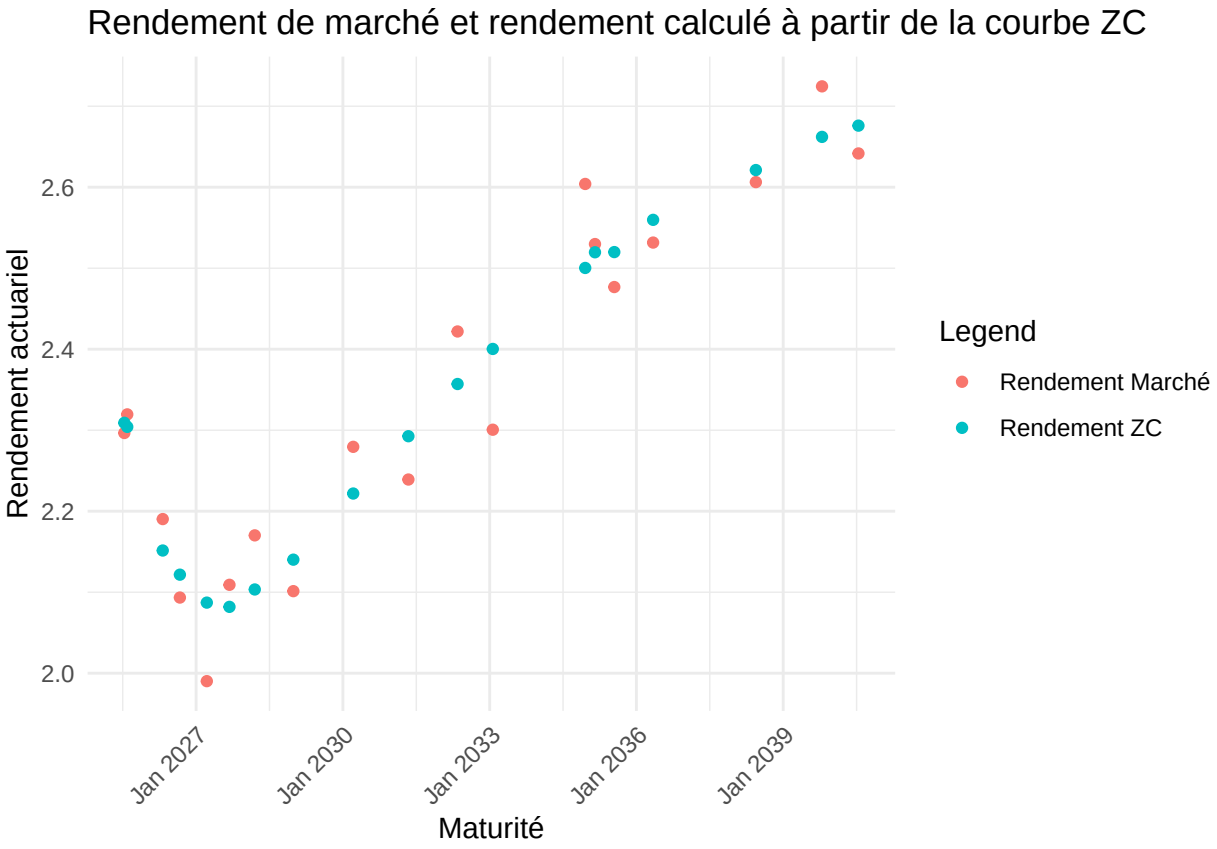
Finalement, on réalise la calibration du modèle.

Le tableau 1 donne les valeurs optimales des paramètres.

Table 1: Swensson parameters fitted to AAA bond prices

β_0	β_1	β_2	β_3	τ_1	τ_2
3.07	-0.61	-2.02	-0.44	1.77	3.75

A titre de vérification, on compare, pour chaque obligation, le taux actuariel de marché au taux calculé à partir de la courbe zéro-coupon.



2 Le modèle de Black-Derman-Toy

On considère le modèle de Black, Derman et Toy décrit dans la note de cours.

La courbe de volatilité (en %) du taux zéro-coupon est donnée par la fonction

```
vol.curve <- splinefun(c(0,5,10,20), c(1, 0.8, 0.6, 0.4)/5)
```

La figure 1 présente sur le même graphique la courbe zéro-coupon et la courbe de volatilité.

Les étapes du calcul:

1. Calibrage de l'arbre: généraliser la méthode de calibration vue en cours pour pouvoir construire un arbre de n pas, et d'incrément Δt.
2. A partir de l'article de Boyle (2000), utiliser les prix d'Arrow-Debreu pour optimiser les calculs.
3. Construire un arbre de maturité 5 ans, par pas de temps de 1 mois.
4. Vérifier la conformité de l'arbre en valorisant une obligation zéro coupon et une obligation à coupon fixe dans l'arbre. Vous devez reproduire exactement les prix obtenus par actualisation des flux avec les taux zéro-coupon.

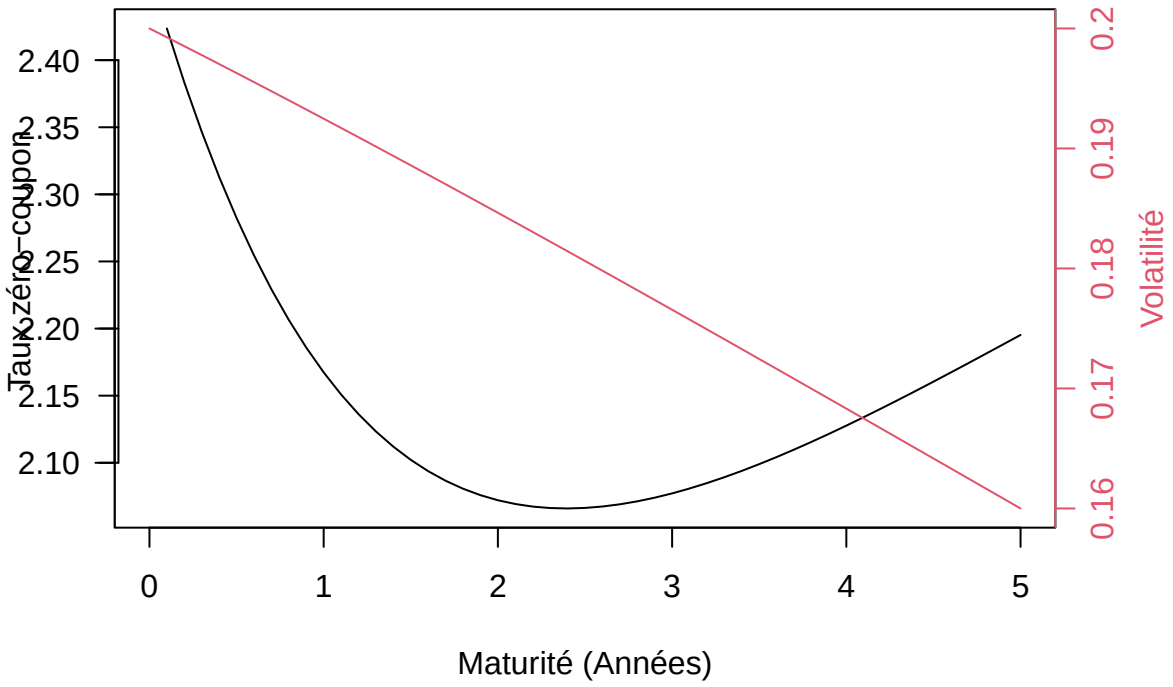


Figure 1: Courbe zéro-coupon et courbe de volatilité

Solution

Calibrage

L'élément de base pour construire un arbre de Black-Derman-Toy est la fonction qui calcule le prix et la volatilité de taux d'une obligation zéro-coupon de maturité $n\Delta t$. On suppose que les variables $\hat{r}(k), \alpha(k), k = 1, \ldots, (n-1)$ sont connues. On note que $\hat{r}(1)$ est obtenu directement à partir de la courbe zero-coupon $z(t)$ et que $\alpha(1) = 1$.

```
PV.Y <- function(bdt.params, n, delta.t) {
  vol = NA
  r.hat <- bdt.params$r.hat
  alpha <- bdt.params$alpha

  # value=1 at time n \Delta t
  pv <- as.vector(rep(1,n+1))
  for(i in seq(n, 1, -1)) {
    iExp <- seq(from=0, to=(i-1), by=1)
    r <- r.hat[i] * exp(2*alpha[i] * iExp)
    discount.fac <- 1/(1+r)^delta.t
    pv <- .5 * discount.fac * pv[2:(i+1)] + .5 * discount.fac * pv[1:i]
    if(i==2) {
      Y.up <- (1/pv[2])^(1/((n-1)*delta.t)) - 1
      Y.down <- (1/pv[1])^(1/((n-1)*delta.t)) - 1
      vol <- (1/2) * log(Y.up / Y.down)
    }
  }
  list(pv=pv[1], vol=vol)
}
```

Définissons le système d'équations à résoudre pour un pas de temps n, sachant que les pas de temps précédents ont été résolus:

```
obj <- function(x, n, delta.t, bdt.params) {
  bdt.params$r.hat[n] <- x[1]
  bdt.params$alpha[n] <- x[2]
  tmp <- PV.Y(bdt.params, n, delta.t)
  res <- numeric(2)
  # z(n) is the zero-coupon yield of a bond maturing at n \Delta t
  res[1] <- tmp$pv - df(n*delta.t)
  res[2] <- tmp$vol - vol.curve(n*delta.t)
  res
  }
```

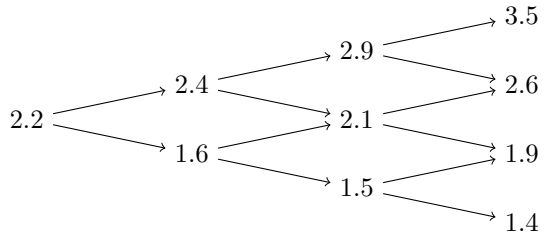
On peut maintenant calibrer l'arbre un pas de temps à la fois, en commençant par la maturité $2\Delta t$. Ici, on construit un arbre de maturité 5 ans par pas de temps de 1 an.

```
bdt.params <- list(r.hat = numeric(5), alpha = numeric(5))

bdt.params$r.hat[1] <- exp(zc.curve(1)/100) - 1
bdt.params$alpha[1] <- 1

for(n in seq(2,5)) {
  # valeurs initiales de r.hat et alpha
  x.0 <- as.vector(c(.02, .01))
  sol <- nleqslv(x.0, obj, n=n, delta.t=1, bdt.params=bdt.params)
  bdt.params$r.hat[n] <- sol$x[1]
  bdt.params$alpha[n] <- sol$x[2]
}
```

Les solutions pour les trois premiers pas de temps sont représentés ci-dessous.



A titre de vérification: on valorise des obligations zero-coupon dans l'arbre, et on doit retrouver les prix calculés directement avec la courbe de taux.

```
delta.t <- 1
error <- 0
for(n in seq(1,5)) {
  P.ZC <- 100 * df(n*delta.t)
  P.BDT <- 100 * PV.Y(bdt.params, n, delta.t)$pv
  print(c(P.ZC, P.BDT))
  error <- max(error, abs(P.ZC-P.BDT))
}
```

```
## [1] 97.85595 97.85595
```

```
## [1] 95.94054 95.94054
## [1] 93.95862 93.95862
## [1] 91.84139 91.84139
## [1] 89.60471 89.60471
```

L'erreur maximale est: 2.5141584×10^{-7}.

Optimisation des calculs

La fonction PV.Y répète inutilement des calculs d'actualisation, alors que le seul élément variable du calcul est l'actualisation du pas de temps $n+1$ vers n à l'aide des variables $\hat{r}(n)$ et $\alpha(n)$. On peut grandement simplifier les calculs en calculant l'actualisation du pas de temps 2 à n avec les prix d'Arrow-Debreu associés aux états de l'étape n.

```
PV.Y.2 <- function(bdt.params, n, delta.t, AD.up, AD.down) {
  vol = NA
  r.hat <- bdt.params$r.hat
  alpha <- bdt.params$alpha
  # value=1 at time n \Delta t
  pv <- as.vector(rep(1,n+1))
  iExp <- seq(from=0, to=(n-1), by=1)
  r <- r.hat[n] * exp(2*alpha[n] * iExp)
  discount.fac <- 1/(1+r)^delta.t
  pv <- .5 * discount.fac * pv[2:(n+1)] + .5 * discount.fac * pv[1:n]
  pv.up <- sum(pv[2:n]*AD.up)
  pv.down <- sum(pv[1:(n-1)]*AD.down)
  Y.up <- (1/pv.up)^(1/((n-1)*delta.t)) - 1
  Y.down <- (1/pv.down)^(1/((n-1)*delta.t)) - 1
  vol <- (1/2) * log(Y.up / Y.down) / sqrt(delta.t)
  pv.0 <- (1/2)*(pv.up+pv.down)/(1+r.hat[1])^delta.t

  list(pv=pv.0, vol=vol)
}
```

La fonction objectif ne change pas, mais il faut mettre à jour les prix d'Arrow-Debreu à chaque itération.

Finalement, ces éléments sont regroupés dans la fonction suivante:

```
BDT.tree <- function(delta.t, horizon) {
  nb.steps <- round(horizon/delta.t)
  bdt.params <- list(r.hat = numeric(nb.steps), alpha = numeric(nb.steps))
  bdt.params$r.hat[1] <- exp(zc.curve(delta.t)/100) - 1
  bdt.params$alpha[1] <- 1
  AD.up <- 1
  AD.down <- 1
  for (n in seq(2, nb.steps)) {
    x.0 <- as.vector(c(bdt.params$r.hat[n-1], bdt.params$alpha[n-1]))
    sol <- nleqslv(x.0, obj.2, n=n, delta.t=delta.t, bdt.params=bdt.params,
                   AD.up=AD.up, AD.down=AD.down)
    if(sol$termcd != 1) {
      print(paste("nleqslv error for n:", n))
      print(sol)
      break
```

```
    }
    bdt.params$r.hat[n] <- sol$x[1]
    bdt.params$alpha[n] <- sol$x[2]
    tmp <- numeric(length=n)
    iExp <- seq(0, (n-1))
    r <- bdt.params$r.hat[n] * exp(2*bdt.params$alpha[n]*iExp)
    # AD prices to up state
    tmp[-1] <- (1/2)*AD.up/(1+r[-1])^delta.t
    tmp[-n] <- tmp[-n] + (1/2) *AD.up/(1+r[-1])^delta.t
    AD.up <- tmp
    tmp <- numeric(length=n)
    tmp[-1] <- (1/2)*AD.down/(1+r[-n])^delta.t
    tmp[-n] <- tmp[-n] + (1/2)* AD.down/(1+r[-n])^delta.t
    AD.down <- tmp
  }
  bdt.params
}
```

Vérification

On vérifie le modèle en valorisant diverses obligations dans l'arbre. Pour simplifier la mise en oeuvre, on suppose que les cash-flows tombent exactement à un incrément dans l'arbre. Pour être précis, il faudrait prendre en compte les dates exactes de paiement et interpoler les valeurs actualisées.

La fonction ci-dessous valorise, dans un arbre Black-Derman-Toy, un montant K payé à une date future $T = N\Delta t$. Cette fonction n'est bien sûr utile qu'à fin de vérification.

```
ZC.discount <- function(bdt.params, delta.t, N, K) {
  r.hat <- bdt.params$r.hat
  alpha <- bdt.params$alpha
  # noeud associé au cash-flow K
  pv <- as.vector(rep(K,N+1))

  for(i in seq(N, 1, -1)) {
    iExp <- seq(from=0, to=i-1, by=1)
    r <- r.hat[i] * exp(2*alpha[i] * iExp)
    discount.fac <- 1/(1+r)^delta.t
    pv <- (discount.fac * pv[2:(i+1)] + discount.fac * pv[1:i])/2
  }
  pv
}
```

```
# Facteur d'actualisation pour un paiement au pas n de l'arbre
df.n <- function(n) {
  df(n*delta.t)
}
```

Calculons de deux manières le prix d'un zero-coupon de maturité 5 ans. Avec un pas de 1 mois, le paiement a lieu à l'étape 60 de l'arbre.

```
delta.t = 1/12
horizon = 10
```

```
bdt.params <- BDT.tree(delta.t, horizon)
n <- 60
K <- 100
PV.BDT <- ZC.discount(bdt.params, delta.t, n, K)
PV.ZC <- 100 * df.n(n)
```

On obtient bien des prix cohérents. Le prix selon la courbe ZC est: 89.605, et le prix selon l'arbre BDT: 89.605.

Pour valoriser une obligation à coupon fixe, on modifie comme suit la fonction d'actualisation dans l'arbre:

```
CF.discount <- function(bdt.params, delta.t, first_index, cf=NULL, pv=NULL) {
  # cash flow dates start at 0
  # BDT parameters (index) start at 1
  r.hat <- bdt.params$r.hat
  alpha <- bdt.params$alpha
  # noeud associé au dernier cash-flow
  if(is.null(pv)) {
    nodes_at_last_step <- tail(cf$dt,1)+1
    pv <- as.vector(rep(tail(cf$flow,1), nodes_at_last_step))
  } else {
    nodes_at_last_step <- length(pv)
  }
  last_index = nodes_at_last_step
  cp <- 0
  for(i_index in seq(last_index-1, first_index, -1)) {
    iExp <- seq(from=0, to=(i_index-1), by=1)
    r <- r.hat[i_index] * exp(2*alpha[i_index] * iExp)
    discount.fac <- 1/(1+r)^delta.t
    pv <- (discount.fac * pv[2:(i_index+1)] + discount.fac * pv[1:i_index])/2
    # on ajoute la valeur du coupon payé à cette date, sauf à la date d'actualisation.
    idx <- which((cf$dt+1) == i_index)
    if(length(idx) == 1) {
      if(i_index > first_index) {
        pv <- pv + cf$flow[idx]
      }  else {
        cp <- cf$flow[idx]
      }
    }
  }
  list(pv=pv, cp=cp)
}
```

Fonction pour construire l'échéancier d'un obligation.

```
bond.cf <- function(maturity, coupon) {
  cf <- list(dt= 12*seq(maturity), flow=rep(coupon, maturity))
  cf$flow[maturity] <- cf$flow[maturity] + 100
  cf
}
```

Valorisation d'une obligation de maturité 5 ans et coupon de 3%: on calcule le prix à l'aide de l'arbre et aussi directement avec les taux zéro-coupon.

```r
cf <- bond.cf(3, 5)
disc.factor <- unlist(lapply(cf$dt, df.n))
PV.ZC <- sum(cf$flow*disc.factor)
PV.BDT <- CF.discount(bdt.params, delta.t, 1, cf=cf)
```

On obtient bien des prix cohérents. Le prix selon la courbe ZC est: 108.346, et le prix selon l'arbre BDT: 108.346.

3 Options sur obligations

3.1 Option sur un zero-coupon

Calculer le prix d'une option d'achat de maturité 1 an, de strike 91, sur une obligation zéro-coupon de maturité 5 ans.

Solution

A maturité de l'option, le zéro-coupon a une maturité résiduelle 4 ans. On calcule la valeur à terme en $T_1 = 12 \times \Delta t$ de l'obligation zéro-coupon.

```r
n = 12
FV <- CF.discount(bdt.params, delta.t, n+1, pv=rep(100, 61))
Strike <- 91
ex <- pmax(FV$pv-Strike, 0)
pv <- CF.discount(bdt.params, delta.t, first_index=1, pv=ex)
```

On obtient un prix de 0.81. On peut obtenir le même résultat en calculant l'espérance de valeur d'exercice, actualisée à la date de calcul:

```r
prob <- dbinom(seq(0,12), 12, 1/2)
pv <- sum(ex*prob)*df(1)
```

Ce qui donne comme prévu la valeur pv = 0.81.

3.2 Options sur obligation

On considère maintenant une obligation de maturité 9 ans versant un coupon annuel de 3%, et une option d'achat de strike 100 et de maturité 1 an sur cette obligation.

Solution

On calcule la valeur à terme en $T_1 = 12 \times \Delta t$ de l'obligation.

```r
cf <- bond.cf(9, 3)
# tranche de temps 13 = 12 mois.
FV <- CF.discount(bdt.params, delta.t, first_index=13, cf=cf)

Strike <- 100
```

```
# valeur d'exercice à maturité
ex <- pmax(FV$pv-Strike, 0)
pv <- CF.discount(bdt.params, delta.t, first_index=1, pv=ex)
```

Pour un strike de 100, l'option d'achat vaut 3.62.

3.3 Obligation avec option de remboursement anticipé

Les obligations avec option de remboursement anticipé ("callable bonds") sont émises de façon régulière. Elles donnent l'opportunité, pour l'émetteur, de pouvoir se refinancer à meilleur compte en cas de baisse des taux.

On considère une obligation de maturité 10 ans, remboursable au pair à la date aniversaire de l'émission, c'est à dire une fois par an, et ce à partir du 5ème aniversaire. Il y aura donc 5 opportunité de remboursement anticipé.

On se propose de valoriser une telle obligation dans le modèle de Black-Derman-Toy afin de mesurer la valeur de cette option de remboursement anticipé.

Pour ce faire, on va calculer la valeur actualisée de l'obligation par récursion inverse dans l'arbre. A chaque date anniversaire où l'option de remboursement peut être exercée, on devra comparer la valeur actualisée de l'obligation à la valeur de remboursement immédiat, et effectuer le choix optimal du point de vue de l'émetteur.

Les détails de l'obligation sont les suivants:

- date de maturité: 23 février 2035
- coupon annuel: 3%

Les obligations sont en grande majorité émises au pair. Quel doit être le taux de coupon pour que ces obligations (avec ou sans clause de remboursement anticipé) puissent être émises au pair?

Solution

On construit l'échéancier de l'obligation

```
maturity <- 10
coupon <- 3
K <- 100
cf <- bond.cf(maturity, coupon)
```

Construction d'un arbre de maturité 10 ans et actualisation: a chaque date anniversaire, on calcule l'exercice optimal, et on ajoute le coupon à la valeur liquidative.

```
bdt.params <- BDT.tree(delta.t, maturity)
pv <- NULL
for(i in seq(9,5)) {
    Val <- CF.discount(bdt.params, delta.t, first_index=i*12+1, cf=cf, pv=pv)
    pv <- pmin(Val$pv, 100)
    pv <- pv + Val$cp
}
```

Finalement on actualise de la 5ème année jusqu'à la date du jour.

```
Val <- CF.discount(bdt.params, delta.t, first_index= 1, cf=cf, pv=pv)
```

Ce qui donne un prix de 102.96. A titre de comparaison, calculons la valeur de l'obligation sans clauses de remboursement anticipé:

```
disc.factor <- unlist(lapply(cf$dt, df.n))
PV.ZC <- sum(cf$flow*disc.factor)
```

Sans l'option de remboursement anticipé, cette même obligation vaudrait 104.361.

Pour déterminer le coupon qui permettrait d'émettre au pair, on résout pour c l'équation $PV(c) = 100$. Avec c le taux de coupon et $P(c)$ la valeur de l'obligation à l'émission.

```
f <- function(coupon) {
  cf <- bond.cf(maturity, coupon)
  pv <- NULL
  for(i in seq(9,5)) {
    Val <- CF.discount(bdt.params, delta.t, first_index=i*12+1, cf=cf, pv=pv)
    pv <- pmin(Val$pv, 100)
    pv <- pv + Val$cp
  }
  Val <- CF.discount(bdt.params, delta.t, first_index= 1, cf=cf, pv=pv)
  Val$pv - 100
}

sol <- uniroot(f, c(1, 4))
```

Ce qui donne un coupon de 2.56 % pour l'obligation remboursable par anticipation. Le même calcul pour l' obligation simple donne:

```
f <- function(coupon) {
  cf <- bond.cf(maturity, coupon)
  disc.factor <- unlist(lapply(cf$dt, df.n))
  100 - sum(cf$flow*disc.factor)
}

sol <- uniroot(f, c(1,4))
```

Ce qui donne un coupon de 2.51 % pour l'obligation simple.