CHAPTER 6

Yield Curve Construction using Cubic Spline

6.1. Cubic Spline Interpolation

In this chapter, we describe a generalized bootstrapping method to determine the yield curve given any available data set of bond prices[1]. In the bootstrapping method as discussed in standard texts, zero-coupon rates are extracted iteratively from the net present value expressions of coupon-bearing bond prices for which discount rates for all coupons are presumably determined from previous steps. With insufficient bond data, such intermediate rates could possibly be missing in the bootstrapping sequence render the entire procedure to cease immediately. Consider, for example, the yield curve construction based on the following sample bond prices.

Bond Price	Face Value	Time to Maturity	Semi-annual Coupon	Zero-coupon Rate
\$98.94	\$100	6 months	\$0	2.13%
\$97.65	\$100	1 year	\$0	2.38%
\$114.26	\$100	2 years	\$5	2.63%

The first two instruments (maturities of six months and one year) are zero-coupon bonds. The corresponding zero-coupon rates of $r_{6\text{-}months} = 2.13\%$ and $r_{1\text{-}year} = 2.38\%$ can be calculated very easily from the bond prices by considering the discount of their face values as

$$\$98.94 = \$100\, e^{-r_{6\text{-}months} \times 0.5} \quad \text{and} \quad \$97.65 = \$100\, e^{-r_{1\text{-}year} \times 1.0}$$

The last instrument is a two-year coupon-bearing bond with coupon payments every six months. In the bootstrapping method, the zero rate $r_{2\text{-}years}$ is calculated from the net present value expression of the two-year bond price given by

$$\$114.26 = \$5\, e^{-r_{6\text{-}months} \times 0.5} + \$5\, e^{-r_{1\text{-}year} \times 1.0} + \$5\, e^{-r_{1.5\text{-}years} \times 1.5} + \$105\, e^{-r_{2\text{-}years} \times 2.0} \qquad (6.1)$$

In equation (6.1), the intermediate zero rate, $r_{1.5\text{-}years}$ for one-and-half years, has not yet been determined in the previous procedure. The two-year zero rate cannot be calculated prior to the determination of such missing rate. Naively, the missing zero rate $r_{1.5\text{-}years}$ can be approximated through linear interpolation between the one-year and two-year zero rates as,

$$r_{1.5\text{-}years} = \tfrac{1}{2}(r_{1\text{-}year} + r_{2\text{-}years})$$

In this way, the two-year zero rate in equation (6.1) can loosely be estimated as $r_{2\text{-}years} = 2.63\%$. In a generalized method, the same problem can be overcome using instead cubic spline interpolation that estimates all missing rates in the bootstrapping procedure.

Spline is a piecewise smooth function joined together by different segments of polynomials. The polynomials of adjacent segments are joined smoothly at break points, called *knots*, with continuous derivatives. Given n knots $\{ (x_1, y_1), (x_2, y_2), \ldots , (x_n, y_n) \}$ as shown in Figure 6.1, interpolation among all these points can be achieved through a spline with $n - 1$ polynomials $\{ \phi_1 , \phi_2 , \ldots, \phi_{n-1} \}$. The simplest solution to the interpolating problem would be the spline with linear polynomials. To improve numerical accuracy using higher order polynomials, additional assumptions beside the smoothness conditions are required to uniquely define the coefficients in the polynomials. A common

[1] See R. Deaves and M. Parlar, "A generalized bootstrap method to determine the yield curve", *Applied Mathematical Finance*, Vol. **7**, No. 4 (2000) p257-270.

choice would be the spline with cubic polynomials defined through additional linear assumptions at both endpoints of the interpolating interval (known as the natural spline conditions).

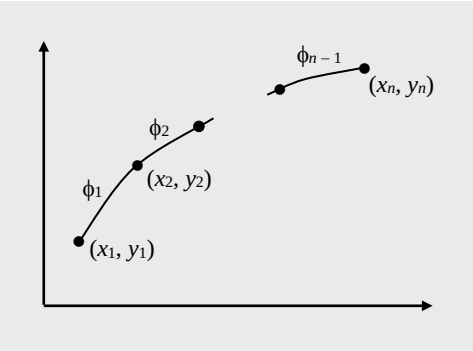


Figure 6.1: Cubic spline interpolation with n knots.

In cubic spline interpolation with n knots at $\{ (x_1, y_1), (x_2, y_2), \ldots , (x_n, y_n) \}$ where $x_1 < x_2 < \ldots < x_n$, there are $n - 1$ cubic polynomials $\phi_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ (for $i = 1, 2, \ldots, n - 1$) with a total of $4(n - 1)$ unknown spline coefficients. The polynomials must generate the corresponding knot values to their left and right ends in each segment as

$$y_i = \phi_i(x_i) \quad , \quad \text{for } i = 1, 2, \ldots, n - 1 \tag{6.2}$$

$$y_{i+1} = \phi_i(x_{i+1}) \quad , \quad \text{for } i = 1, 2, \ldots, n - 1 \tag{6.3}$$

The smoothness conditions are given by the continuities of the polynomials as well as their first and second derivatives, $\phi_i'(x) = b_i + 2c_i x + 3d_i x^2$ and $\phi_i''(x) = 2c_i + 6d_i x$ respectively, at each knot. The polynomials are continuous at each knot according to equations (6.2) and (6.3). The continuities of their first and second derivatives are given by

$$\phi_i'(x_{i+1}) = \phi_{i+1}'(x_{i+1}) \quad , \quad \text{for } i = 1, 2, \ldots, n - 2 \tag{6.4}$$

$$\phi_i''(x_{i+1}) = \phi_{i+1}''(x_{i+1}) \quad , \quad \text{for } i = 1, 2, \ldots, n - 2 \tag{6.5}$$

There are all together only $4(n - 1) - 2$ matching conditions as expressed in equations (6.2) to (6.5). The spline coefficients can be uniquely defined through the natural spline conditions that force the spline to be linear outside the interpolating interval. This is given by the vanishing of second derivatives at both end knots as

$$\phi_1''(x_1) = \phi_{n-1}''(x_n) = 0 \tag{6.6}$$

The matching conditions together with the natural spline conditions can be rewritten in matrix representation as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_{n-1} \\ y_2 \\ \vdots \\ y_n \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix} = \boldsymbol{M} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ \vdots \\ \vdots \\ a_{n-1} \\ b_{n-1} \\ c_{n-1} \\ d_{n-1} \end{pmatrix} \tag{6.7}$$

There are $4(n-1)$ rows in the two column vectors of (6.7) associated with the same number of conditions as expressed in the above equations. The $4(n-1) \times 4(n-1)$ square matrix $\boldsymbol{M}$ is shown in Figure 6.2. As indicated in the figure, the highlighted entries can easily be read off from the conditions in (6.2) to (6.6) while all other entries are zero. The spline coefficients on the right side of (6.7) can be determined by inverting the matrix equation with the inverse of matrix $\boldsymbol{M}$.

***Example* 6.1** :

Consider the cubic spline interpolation for the zero-coupon rates as extracted previously based on the sample bond prices. In this case, we have $n = 3$ knots located at

$$\begin{aligned} x_1 &= 0.5 \text{ year} \quad , \; y_1 = 2.13\% \\ x_2 &= 1.0 \text{ year} \quad , \; y_2 = 2.38\% \\ x_3 &= 2.0 \text{ years} \; , \; y_3 = 2.63\% \end{aligned}$$

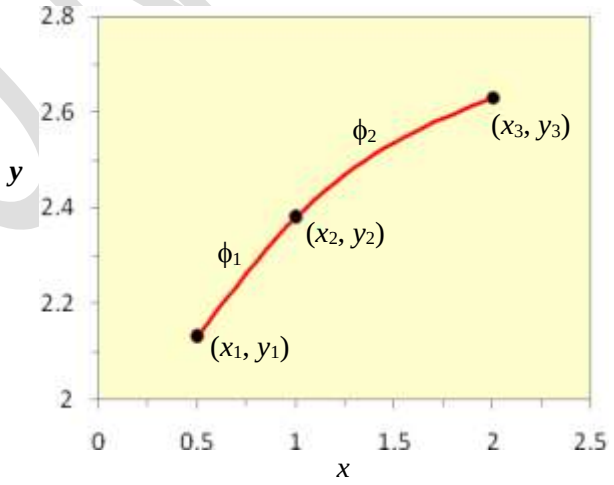
It requires two cubic polynomials $\phi_1(x)$ and $\phi_2(x)$ in the interpolation. Their coefficients can be determined by solving the matrix equation in (6.7) as

$$\begin{pmatrix} 2.13 \\ 2.38 \\ 2.38 \\ 2.63 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0.5 & 0.25 & 0.125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 \\ 0 & 1 & 2 & 3 & 0 & -1 & -2 & -3 \\ 0 & 0 & 2 & 6 & 0 & 0 & -2 & -6 \\ 0 & 0 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 12 \end{pmatrix} \begin{pmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{pmatrix}$$

The cubic spline is then calculated to be

$$\phi_1(x) = 1.88 + 0.4167\,x + 0.25\,x^2 - 0.1667\,x^3 \quad , \text{ for } \; 0.5 \le x \le 1.0$$

$$\phi_2(x) = 1.63 + 1.1667\,x - 0.50\,x^2 + 0.0833\,x^3 \quad , \text{ for } \; 1.0 \le x \le 2.0$$



We want to develop first a generic VBA routine called CubicSpline( ) that implements the cubic spline interpolation by solving the coefficients in equation (6.7) given knots. It would be useful in the construction of yield curve under the problem of missing rates. The pseudo code of CubicSpline( ) is given by Code 6.1. It reads in the location of n knots at $\{ x_1, \ldots, x_n \}$ and $\{ y_1, \ldots, y_n \}$, and returns the spline coefficients $\{ a_1, \ldots, a_{n-1} \}$, $\{ b_1, \ldots, b_{n-1} \}$, $\{ c_1, \ldots, c_{n-1} \}$, and $\{ d_1, \ldots, d_{n-1} \}$ in the interpolation. The VBA code of CubicSpline( ) is given by Code 6.2. For the calculation of the spline coefficients, we have again used the routine SolveAxb( ) to invert the matrix equation in (6.7).

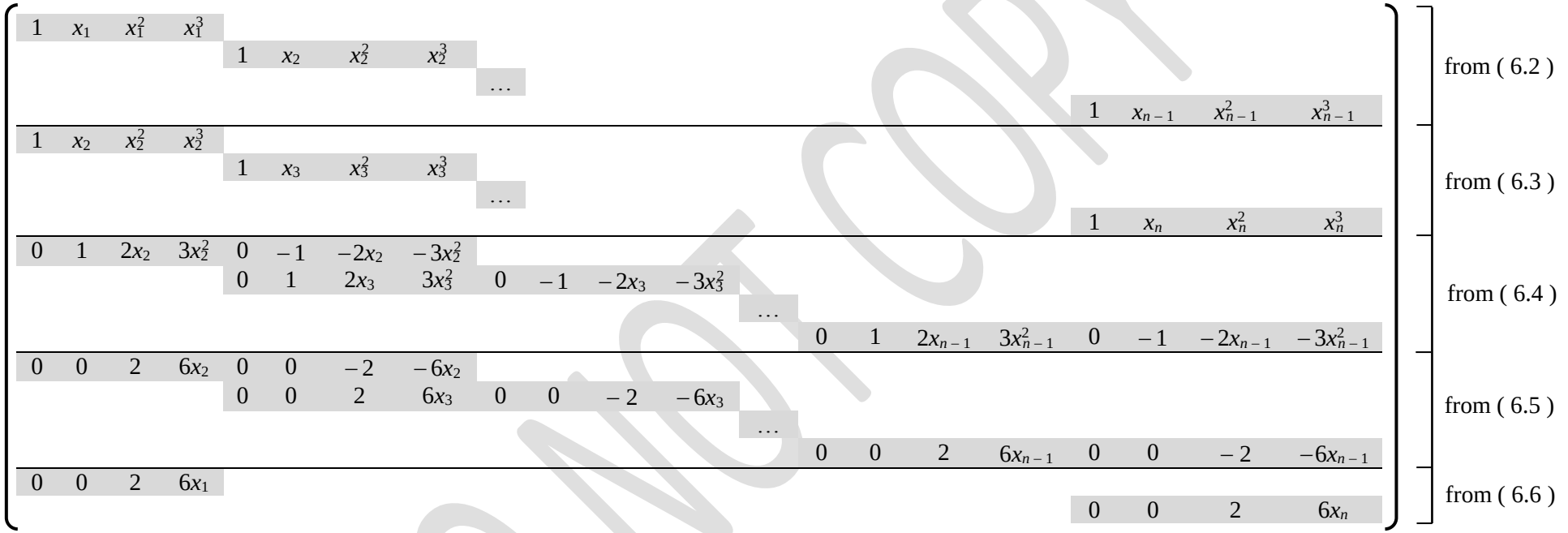


Figure 6.2: The square matrix $\boldsymbol{M}$ of size $4(n-1) \times 4(n-1)$. The undepicted entries are all zero.

CubicSpline(n , $x(1:n)$, $y(1:n)$, $a(1:n-1)$, $b(1:n-1)$, $c(1:n-1)$, $d(1:n-1)$)

# define the column vector $\boldsymbol{R}$ on the left side of equation (6.7)
For ($i = 1$ to $n-1$) { $R(i) = y(i)$
$R(n-1+i) = y(i+1)$
$R(2(n-1)+i) = 0$
$R(3(n-1)+i) = 0$ }

# initialize the entries of matrix $\boldsymbol{M}$
For ($i = 1$ to $4(n-1)$) { For ($j = 1$ to $4(n-1)$) { $M(i, j) = 0$ } }

define the entries in the first ($n-1$) rows
$ptr = 0$, For ($i = 1$ to $n-1$) { $M(ptr+i, 4(i-1)+1) = 1$
$M(ptr+i, 4(i-1)+2) = x(i)$
$M(ptr+i, 4(i-1)+3) = x^2(i)$
$M(ptr+i, 4(i-1)+4) = x^3(i)$ }
define the entries in the second ($n-1$) rows
$ptr = n-1$, For ($i = 1$ to $n-1$) { $M(ptr+i, 4(i-1)+1) = 1$
$M(ptr+i, 4(i-1)+2) = x(i+1)$
$M(ptr+i, 4(i-1)+3) = x^2(i+1)$
$M(ptr+i, 4(i-1)+4) = x^3(i+1)$ }
define the entries in the following ($n-2$) rows
$ptr = 2(n-1)$, For ($i = 1$ to $n-2$) { $M(ptr+i, 4(i-1)+2) = 1$
$M(ptr+i, 4(i-1)+3) = 2x(i+1)$
$M(ptr+i, 4(i-1)+4) = 3x^2(i+1)$
$M(ptr+i, 4(i-1)+6) = -1$
$M(ptr+i, 4(i-1)+7) = -2x(i+1)$
$M(ptr+i, 4(i-1)+8) = -3x^2(i+1)$ }
define the entries in the next ($n-2$) rows
$ptr = 3(n-1)-1$, For ($i = 1$ to $n-2$) { $M(ptr+i, 4(i-1)+3) = 2$
$M(ptr+i, 4(i-1)+4) = 6x(i+1)$
$M(ptr+i, 4(i-1)+7) = -2$
$M(ptr+i, 4(i-1)+8) = -6x(i+1)$ }
define the entries in the last 2 rows
$ptr = 4(n-1)-2$
$M(ptr+1, 3) = 2$, $M(ptr+1, 4) = 6x(1)$
$M(ptr+2, 4(n-1)-1) = 2$, $M(ptr+2, 4(n-1)) = 6x(n)$

determine the spline coefficients $\boldsymbol{Q}$ by solving the matrix equation
$\boldsymbol{Q}(1:4(n-1)) = \boldsymbol{M}^{-1}(1:4(n-1), 1:4(n-1))\ \boldsymbol{R}(1:4(n-1))$

For ($i = 1$ to $n-1$) { $a(i) = Q(4(i-1)+1)$
$b(i) = Q(4(i-1)+2)$
$c(i) = Q(4(i-1)+3)$
$d(i) = Q(4(i-1)+4)$ }

Code 6.1: Pseudo code of the CubicSpline( ) routine

```
Sub CubicSpline(n As Integer, x() As Double, y() As Double, ByRef a() As Double, ByRef b() As Double, ByRef c() As Double,
                ByRef d() As Double)

   Dim i As Integer, j As Integer, ptr As Integer
   Dim Mmatrix() As Double: ReDim Mmatrix(1 To 4 * (n - 1), 1 To 4 * (n - 1))
   Dim Rvec() As Double: ReDim Rvec(1 To 4 * (n - 1))
   Dim Qvec() As Double: ReDim Qvec(1 To 4 * (n - 1))

   For i = 1 To (n - 1)
      Rvec(i) = y(i)
      Rvec(n - 1 + i) = y(i + 1)
      Rvec(2 * (n - 1) + i) = 0
      Rvec(3 * (n - 1) + i) = 0
   Next i

   For i = 1 To 4 * (n - 1)
      For j = 1 To 4 * (n - 1): Mmatrix(i, j) = 0: Next j
   Next i

   ptr = 0
   For i = 1 To (n - 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 1) = 1
      Mmatrix(ptr + i, 4 * (i - 1) + 2) = x(i)
      Mmatrix(ptr + i, 4 * (i - 1) + 3) = x(i) ^ 2
      Mmatrix(ptr + i, 4 * (i - 1) + 4) = x(i) ^ 3
   Next i

   ptr = n - 1
   For i = 1 To (n - 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 1) = 1
      Mmatrix(ptr + i, 4 * (i - 1) + 2) = x(i + 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 3) = x(i + 1) ^ 2
      Mmatrix(ptr + i, 4 * (i - 1) + 4) = x(i + 1) ^ 3
   Next i

   ptr = 2 * (n - 1)
   For i = 1 To (n - 2)
      Mmatrix(ptr + i, 4 * (i - 1) + 2) = 1
      Mmatrix(ptr + i, 4 * (i - 1) + 3) = 2 * x(i + 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 4) = 3 * x(i + 1) ^ 2
      Mmatrix(ptr + i, 4 * (i - 1) + 6) = -1
      Mmatrix(ptr + i, 4 * (i - 1) + 7) = -2 * x(i + 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 8) = -3 * x(i + 1) ^ 2
   Next i

   ptr = 3 * (n - 1) - 1
   For i = 1 To (n - 2)
      Mmatrix(ptr + i, 4 * (i - 1) + 3) = 2
      Mmatrix(ptr + i, 4 * (i - 1) + 4) = 6 * x(i + 1)
      Mmatrix(ptr + i, 4 * (i - 1) + 7) = -2
      Mmatrix(ptr + i, 4 * (i - 1) + 8) = -6 * x(i + 1)
   Next i

   ptr = 4 * (n - 1) - 2
   Mmatrix(ptr + 1, 3) = 2
   Mmatrix(ptr + 1, 4) = 6 * x(1)
   Mmatrix(ptr + 2, 4 * (n - 1) - 1) = 2
   Mmatrix(ptr + 2, 4 * (n - 1)) = 6 * x(n)

   Call SolveAxb(Mmatrix(), Rvec(), Qvec(), 4 * (n - 1), 1, 1, 1)

   For i = 1 To (n - 1)
      a(i) = Qvec(4 * (i - 1) + 1)
      b(i) = Qvec(4 * (i - 1) + 2)
      c(i) = Qvec(4 * (i - 1) + 3)
      d(i) = Qvec(4 * (i - 1) + 4)
   Next i

End Sub
```

Code 6.2: VBA code of the CubicSpline( ) routine

6.2. Yield Curve Construction

In example 6.1, we have adopted a two-year zero rate of $r_{2\text{-}years} = 2.63\%$ based on a linear approximation. In this case, the two-year bond price in equation (6.1) will not necessary be satisfied with the interpolated one-and-half year zero rate $r_{1.5\text{-}years} = \phi_2(1.5)$ using the cubic spline. In a generalized method of yield curve construction, $r_{2\text{-}years}$ should be determined such that the resulting cubic spline will provide an interpolated value of $r_{1.5\text{-}years}$ that exactly reproduces the two-year bond price. This can be achieved through a numerical search of $r_{2\text{-}years}$ using the Newton-Raphson procedure over the error on the net present value expression as

$$\$114.26 = \$5\, e^{-r_{6\text{-}months} \times 0.5} + \$5\, e^{-r_{1\text{-}year} \times 1.0} + \$5\, e^{-\phi_2(1.5 \mid r_{2\text{-}years}) \times 1.5} + \$105\, e^{-r_{2\text{-}years} \times 2.0} \qquad (6.8)$$

where $\phi_2(x \mid r_{2\text{-}years})$ denotes the resulting cubic polynomial for a specific value of $r_{2\text{-}years}$. As described in the VBA coding below, we can simply call the NewtonRaphson( ) routine in previous chapter with one variable and initiate the search using the value of the nearest zero rate. The error in (6.8) can be calculated based on the following setup in FunctionArray( ). Under the precision requirement of 10^{-8}, the two-year zero rate is extracted to be $r_{2\text{-}years} = 2.63125423\%$ with a slight correction from the linear approximation.

```
Sub Searchr2y()
  Dim r2y(1 To 1) As Double, prec As Double, precFlag As Boolean, maxDev As Double
  r2y(1) = 2.38
  prec = 0.00000001
  Call NewtonRaphson(1, prec, r2y, precFlag, maxDev)
  Range("B2").Value = r2y(1)
  Range("B3").Value = precflag
  Range("B4").Value = maxDev
End Sub

Sub FunctionArray(n As Integer, r2y() As Double, ByRef NPVerr() As Double)
  Dim x(1 To 3) As Double, y(1 To 3) As Double
  Dim a(1 To 2) As Double, b(1 To 2) As Double, c(1 To 2) As Double, d(1 To 2) As Double
  x(1) = 0.5
  x(2) = 1.0
  x(3) = 2.0
  y(1) = 2.13
  y(2) = 2.38
  y(3) = r2y(1)
  Call CubicSpline(3, x, y, a, b, c, d)
  Dim xm As Double: xm = 1.5
  Dim ym As Double: ym = a(2) + b(2) * xm + c(2) * xm ^ 2 + d(2) * xm ^ 3
  NPVerr(1) = 114.26 - 5 * Exp(-(y(1) / 100) * x(1)) - 5 * Exp(-(y(2) / 100) * x(2)) - 5 * Exp(-(ym / 100) * xm) - 105 * Exp(-(y(3) / 100) * x(3))
End Sub
```

Practically, we perform a yield curve construction based on a set of observed bond prices { $B_1, B_2, \ldots, B_n$ } with terms to maturity { $T_1 < T_2 < \ldots < T_n$ } and face values { $L_1, L_2, \ldots, L_n$ }. Associated with each of these bonds, there are fixed coupon payments { $c_1 = 0, c_2, \ldots, c_n$ } scheduled sequentially in time at

$$t_{21}, t_{22}, \ldots \le T_2 \text{ , for bond } B_2$$

$$\ldots\ \ldots$$

$$t_{n1}, t_{n2}, \ldots \le T_n \text{ , for bond } B_n$$

We denote $r(t)$ to be the zero-coupon rate with term to maturity t. The objective is to extract the zero rates { $r(T_1), \ldots, r(T_n)$ } according to the observed bond prices. For the construction to be attainable, the first bond B_1 in the data set with the shortest term must be zero coupon, and all of the above coupon payments must be scheduled on or after T_1. The net present value expressions for each of these bonds are given by

$$B_1 = L_1 \, e^{-r(T_1) \, T_1} \tag{6.9}$$

$$B_2 = c_2 \, e^{-r(t_{21}) \, t_{21}} + c_2 \, e^{-r(t_{22}) \, t_{22}} + \ldots + (\, c_2 + L_2 \,) \; e^{-r(T_2) \, T_2} \quad , \quad T_1 \le t_{21} < t_{22} \; < \ldots \le T_2$$

$$\ldots \ldots$$

$$B_n = c_n \, e^{-r(t_{n1}) \, t_{n1}} + c_n \, e^{-r(t_{n2}) \, t_{n2}} + \ldots + (\, c_n + L_n \,) \; e^{-r(T_n) \, T_n} \quad , \quad T_1 \le t_{n1} < t_{n2} < \ldots \le T_n$$

In equation (6.9), the bootstrapping procedure starts off from the determination of the shortest zero rate $r(T_1)$ from B_1. However, it is clear that there are possibly missing discount rates for the coupon payments in the immediate determination of $r(T_2)$, and similarly for all others in the following steps, from the coupon-bearing bond prices.

In the generalized method, such problem can be overcome using cubic spline interpolation with knots at $\{ T_1, T_2, \ldots, T_n \}$. Discount rates for coupon payments in (6.9) can be estimated by their interpolated values based on $n - 1$ cubic polynomials $\{ \phi_1, \phi_2, \ldots, \phi_{n-1} \}$ in the spline as

$$r(t) = \phi_k(\, t \mid r(T_1), \ldots, r(T_n) \,) \quad , \quad T_k < t \le T_{k+1} \tag{6.10}$$

In equation (6.10), $\phi_k(\, t \mid r(T_1), \ldots, r(T_n) \,)$ denotes the resulting cubic polynomials using (6.7) with specific knot values $\{ r(T_1), \ldots, r(T_n) \}$. Thus, we need to determine the zero rates $\{ r(T_1), \ldots, r(T_n) \}$ such that the resulting cubic spline will generate interpolated values of coupon discount rates consistent with the bond prices. This is similar to the problem in (6.8) and can also be achieved through a numerical search using the Newton-Raphson procedure over the errors on the expressions in (6.9). As B_1 is assumed to be a zero-coupon bond, the shortest zero rate $r(T_1)$ can be extracted directly from (6.9) as

$$r(T_1) = -\frac{1}{T_1} \, ln\left[\frac{B_1}{L_1} \right] \tag{6.11}$$

However, it is convenient to include $r(T_1)$ in the Newton-Raphson procedure and formulate a search for the entire set of zero rates with (6.11) taken to be the common initial value. It can be shown that $r(T_1)$ will remain stationary during the search for which the extra loading on the procedure is insignificant.

6.3. EXCEL Plus VBA Implementation

The errors on the net present value expressions in equation (6.9) can be calculated based on the FunctionArray( ) routine with pseudo code given by Code 6.3. Given zero rates $\{ r(T_1), \ldots, r(T_n) \}$, the routine will return an array of error values $\{ g_1, \ldots, g_n \}$ with respect to different expressions in (6.9). Bond data such as bond prices, terms to maturity, face values, coupon payments, number of coupons, and the payment schedules are inputed from EXCEL and stored as VBA arrays. The layout of the data interface in EXCEL will be discussed later in this section. For specific values of the zero rates during the search, the corresponding cubic spline is first calculated by calling the CubicSpline( ) routine. In Code 6.3, $t_c(i, j)$ denotes the time of the j-th coupon payment for the i-th bond. The double loop with labels i and j will run through all coupon payments for the entire set of coupon-bearing bonds. The interpolated value of the discount rate with term $t_c(i, j)$ can be determined according to (6.10) by identifying the relevant segment from T_k to T_{k+1} in the spline such that $T_k < t_c(i, j) \le T_{k+1}$. The interpolation can then be performed using the cubic polynomial in this segment parameterized by the spline coefficients a_k, b_k, c_k, and d_k. To determine the required value of k, it is straight forward to run k from 1 to $i - 1$ and recall that $t_c(i, j) \le T_i$ as defined in (6.9). In Code 6.3, we have made efficient

use of a pointer *ptr* that records the location of T_k (setting $ptr = k$) for previous coupon payment. In this way, we can instead run k from *ptr* to $i - 1$ and start off the pointer from $ptr = 1$ at every initial coupon payment. Once we have identified the appropriate value of k, we should immediately proceed to the next coupon payment with new value of j. It should be noted that the k-loop does not provide the checking for $t_c(i, j)$ at exactly the leftmost knot T_1. We have included before the k-loop such checking and assigning the knot value of $r_c(i , j) = r(T_1)$. As discussed in earlier chapter, the checking should be conducted through the absolute limit $| t_c(i, j) - T_1 | \le \varepsilon$ with precision $\varepsilon = 10^{-14}$ to avoid possible floating point problem. Using the interpolated values of the coupon discount rates, the errors on the net present value expressions in (6.9) can then be calculated very easily in terms of an array $\{ g_1, \ldots , g_n \}$.

The numerical search of the zero rates can be conducted very efficiently using the CalZeroRates( ) routine with pseudo code as depicted in Code 6.4. The number of unknown zero rates n and the designated precision limit of the search *prec* are firstly defined in the routine. Data on the shortest zero-coupon bond are also inputed from EXCEL as the numerical search should be initiated from the value defined in (6.11). The search can be performed by calling the NewtonRaphson( ) routine that will in turn call the FunctionArray( ) routine in Code 6.3 for the evaluation of the objective functions $\{ g_1, \ldots , g_n \}$. Upon exit from a successful search, it will return the zero rates $\{ r(T_1), \ldots , r(T_n) \}$ for which all error values $\{ g_1, \ldots , g_n \}$ have vanished under the designated precision limit with *precflag* = TRUE and $maxdev \le prec$. The returned zero rates together with the precision flag and the maximum deviation should then be outputted into EXCEL.

Figure 6.3 illustrates the layout of the bond data interface in EXCEL spreadsheet[2]. Information on different bonds is arranged in row-by-row format starting from the ninth row. These include term to maturity (column A), bond price (column C), face value (column D), coupon payment (column E), number of coupons (column F), and payment schedule (column G onward). The VBA code of the FunctionArray( ) routine is given by Code 6.5. Accordingly, bond data are read off row-by-row from spreadsheet with the use of the **OFFSET** function relative to the header cells in the eighth row. The VBA code of the main CalZeroRates( ) routine is given by Code 6.6. It can be invoked through the "Calculate Zero rates" button in EXCEL. The total number of bonds in the data and the designated precision of the search are read off from the named cells B4(nbond) and E4(prec), respectively. Initial values of the zero rates are defined based on the shortest zero-coupon bond at row offset of one from the header. Upon exit from the Newton-Raphson procedure, the returned zero rates are outputted into spreadsheet underneath the header "Zero Rate" and adjacent to the corresponding terms to maturity. As reference, the precision flag and the maximum deviation of the search are also outputted into cells E5 and E6, respectively.

	A	B	C	D	E	F	G	H	I	J	K
1											
2	**Bond Data :**										
3											
4	Number of Bonds	8	Calculate Zero Rates	Precision	1.00E-08						
5				Flag	TRUE						
6				Max. Dev	5.34E-14						
7											
8	Term to Maturity	Zero Rate	Bond Price	Face Value	Coupon Payment	Number of Coupons	Payment Schedule				
9	0.083	4.448%	99.63	100.00	0.000	0					
10	0.250	4.546%	98.87	100.00	0.000	0					
11	0.500	4.633%	97.71	100.00	0.000	0					
12	1.000	4.783%	95.33	100.00	0.000	0					
13	2.000	4.933%	100.00	100.00	2.495	4	0.500	1.000	1.500	2.000	
14	4.000	5.055%	100.00	100.00	2.555	8	0.500	1.000	1.500	2.000	2.500
15	7.500	5.192%	100.00	100.00	2.620	15	0.500	1.000	1.500	2.000	2.500
16	18.000	4.233%	100.00	100.00	2.250	36	0.500	1.000	1.500	2.000	2.500
17											
18											
19											
20	Scale : yearly										

Figure 6.3: Bond data interface in EXCEL spreadsheet

[2] Refer to yield_curve.xls

As reference, it is useful to generate a smooth yield curve and display in spreadsheet the zero rates for intermediate values of the maturity term. This can be done by calling the CubicSpline( ) routine with the bootstrapped zero rates from CalZeroRates( ) as knot values. The VBA code of the GenYieldCurve( ) routine that performs this task is given by Code 6.7. It can be invoked through the "Generate Yield Curve" button in EXCEL. Bootstrapped zero rates and the corresponding terms to maturity are inputted into VBA through row offset from the header cells B8 and A8, respectively. They are taken as knots for the CubicSpline( ) routine in order to generate the cubic spline approximation of the yield curve. As shown in Figure 6.4, we have defined in the named cell E23(npoint) the number of internal points between any two adjacent knots T_i and T_{i+1}. Subsequent values of the maturity term from the left to right ends are defined through the double-loop with labels i and j as

$$term = T_i + j \times (T_{i+1} - T_i)/(npoint + 1) \ , \ \text{where} \ \ i = 1, \ldots, n-1 \ \ \text{and} \ \ j = 0, \ldots, npoint$$

In this way, a smooth yield can be generated using the interpolated value $\phi_i(term)$ and outputted into spreadsheet underneath the header cells A24 and B24. It should be noted that the right-end knot at T_n has not been covered by the double loop. For completeness, it has been appended to the yield curve at the end of Code 6.7.

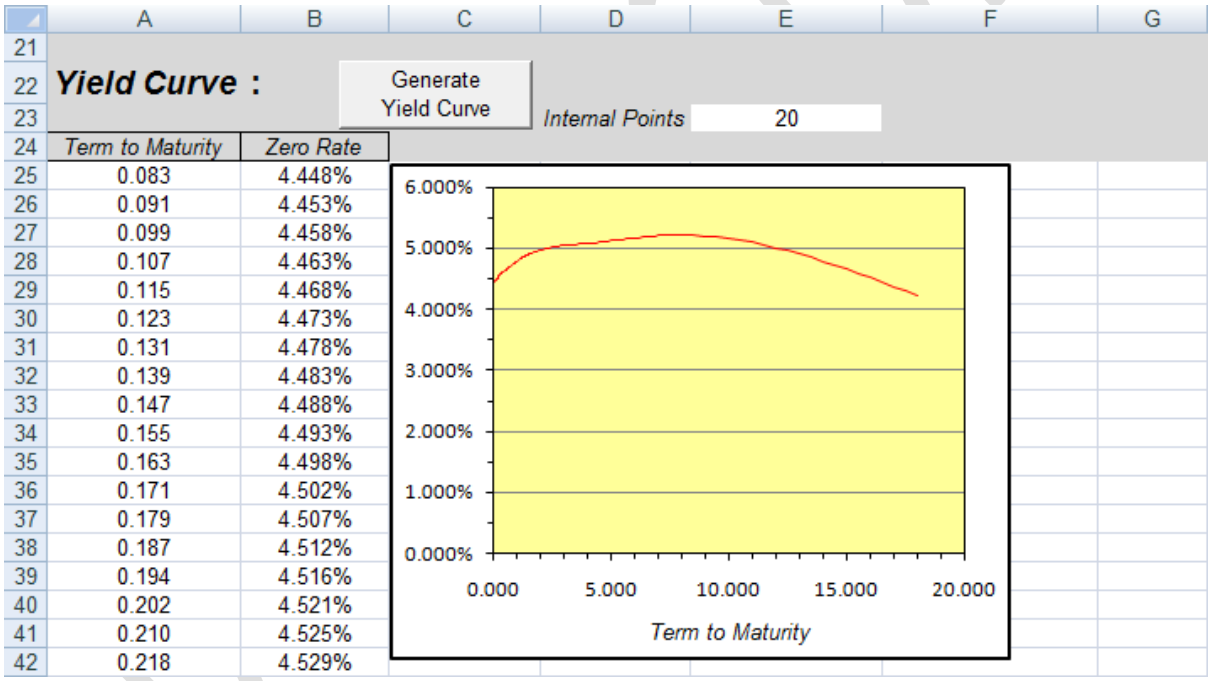


Figure 6.4: Constructed yield curve in EXCEL spreadsheet

FunctionArray(n , $r(1:n)$, $g(1:n)$)

input bond data from Excel
Read terms to maturity { $T_1, T_2, \ldots, T_n$ } as array $T(1:n)$
Read bond prices { $B_1, B_2, \ldots, B_n$ } as array $B(1:n)$
Read face values { $L_1, L_2, \ldots, L_n$ } as array $L(1:n)$
Read coupon payments { $c_1 = 0, c_2, \ldots, c_n$ } as array $C(1:n)$
Read number of coupon for each bond{ $m_1 = 0, m_2, \ldots, m_n$ } as array $m(1:n)$
Read coupon payment schedule { { $t_{21}, t_{22}, \ldots$ }, … , { $t_{n1}, t_{n2}, \ldots$ } } as array $t_c(1:n, 1:\text{Max}(m))$

generate the cubic spline given knot values rs at Ts
Call *CubicSpline*(n , $T(1:n)$, $r(1:n)$, $a(1:n-1)$, $b(1:n-1)$, $c(1:n-1)$, $d(1:n-1)$)

# interpolate the discount rates r_c at all coupon payments t_c according to (6.10)
For($i = 2$ to n) {
  $ptr = 1$
  For($j = 1$ to $m(i)$) {
    $\tau = t_c(i, j)$
    If($|\tau - T(1)| \le \varepsilon$){ $r_c(i, j) = r(1)$, Next j }
    For($k = ptr$ to $i - 1$) { If($T(k) < \tau \le T(k+1)$) Then
      $r_c(i, j) = a(k) + b(k)\,\tau + c(k)\,\tau^2 + d(k)\,\tau^3$
      $ptr = k$
      Next j
    Endif }
  }
}

calculate the NPV errors g according to (6.9)

For($i = 1$ to n) { $g(i) = B(i) - L(i)\, e^{-r(i)\,T(i)}$
  For($j = 1$ to $m(i)$) { $g(i) = g(i) - C(i)\, e^{-r_c(i,j)\,t_c(i,j)}$ }
}

Code 6.3: Pseudo code of the FunctionArray( ) routine.

CalZeroRates( )

input from EXCEL number of unknown zero rates
Read n

input from EXCEL the designated precision limit of the numerical search
Read *prec*

input from EXCEL data on the shortest zero-coupon bond and initiate the search according to (6.11)
Read T_1, B_1, and L_1
$r_{init} = -(1/T_1)\, ln(B_1 / L_1)$
For($i = 1$ to n) { $r(i) = r_{init}$ }

perform the numerical search for the zero rates using Newton-Raphson procedure
Call *NewtonRaphson*(n , *prec* , $r(1 : n)$, *precflag* , *maxdev*)

output to EXCEL the returned zero rates, precision flag, and maximum deviation
Write $r(1 : n)$, *precflag*, and *maxdev*

Code 6.4: Pseudo code of the CalZeroRates( ) routine.

```
Option Explicit
Private Const eps = 1 * 10 ^ -14
Private Const mmax = 100

Sub FunctionArray(n As Integer, rzero() As Double, ByRef g() As Double)

  Dim T() As Double: ReDim T(1 To n)
  Dim Bprice() As Double: ReDim Bprice(1 To n)
  Dim par() As Double: ReDim par(1 To n)
  Dim coupon() As Double: ReDim coupon(1 To n)
  Dim m() As Integer: ReDim m(1 To n)
  Dim tc() As Double: ReDim tc(1 To n, 1 To mmax)
  Dim rc() As Double: ReDim rc(1 To n, 1 To mmax)

  Dim i As Integer, j As Integer, k As Integer, ptr As Integer
  Dim tau As Double
  Dim a() As Double: ReDim a(1 To n - 1)
  Dim b() As Double: ReDim b(1 To n - 1)
  Dim c() As Double: ReDim c(1 To n - 1)
  Dim d() As Double: ReDim d(1 To n - 1)

  'input bond data
  For i = 1 To n
    T(i) = Range("A8").Offset(i, 0)
    Bprice(i) = Range("C8").Offset(i, 0)
    par(i) = Range("D8").Offset(i, 0)
    coupon(i) = Range("E8").Offset(i, 0)
    m(i) = Range("F8").Offset(i, 0)
    For j = 1 To m(i)
      tc(i, j) = Range("G8").Offset(i, j - 1)
    Next j
  Next i

  'generate the cubic spline
  Call CubicSpline(n, T, rzero, a, b, c, d)

  'interpolate the coupon discount rates
  For i = 2 To n
    ptr = 1
    For j = 1 To m(i)
      tau = tc(i, j)
      If (Abs(tau - T(1)) <= eps) Then
        rc(i, j) = rzero(1)
        GoTo nextj
      End If
      For k = ptr To i - 1
        If (tau > T(k) And tau <= T(k + 1)) Then
          rc(i, j) = a(k) + b(k) * tau + c(k) * tau ^ 2 + d(k) * tau ^ 3
          ptr = k
          GoTo nextj
        End If
      Next k
nextj: Next j
  Next i

  'calculate the NPV errors
  For i = 1 To n
    g(i) = Bprice(i) - par(i) * Exp(-rzero(i) * T(i))
    For j = 1 To m(i)
      g(i) = g(i) - coupon(i) * Exp(-rc(i, j) * tc(i, j))
    Next j
  Next i

End Sub
```

Code 6.5: VBA code of the FunctionArray( ) routine.

```
Sub CalZeroRates()

    Dim n As Integer: n = Range("nbond").Value
    Dim prec As Double: prec = Range("prec").Value

    Dim T As Double: T = Range("A8").Offset(1, 0)
    Dim Bprice As Double: Bprice = Range("C8").Offset(1, 0)
    Dim par As Double: par = Range("D8").Offset(1, 0)
    Dim rinit As Double: rinit = -(1 / T) * Log(Bprice / par)

    Dim i As Integer
    Dim rzero() As Double: ReDim rzero(1 To n)
    Dim precFlag As Boolean
    Dim maxDev As Double

    For i = 1 To n: rzero(i) = rinit: Next i

    Call NewtonRaphson(n, prec, rzero, precFlag, maxDev)

    For i = 1 To n: Range("B8").Offset(i, 0) = rzero(i): Next i
    Range("E5").Value = precFlag
    Range("E6").Value = maxDev

End Sub
```

Code 6.6: VBA code of the CalZeroRates( ) routine.

```
Sub GenYieldCurve()

    Dim n As Integer: n = Range("nbond").Value
    Dim T() As Double: ReDim T(1 To n)
    Dim rzero() As Double: ReDim rzero(1 To n)
    Dim i As Integer, j As Integer, k As Integer

    Dim a() As Double: ReDim a(1 To n - 1)
    Dim b() As Double: ReDim b(1 To n - 1)
    Dim c() As Double: ReDim c(1 To n - 1)
    Dim d() As Double: ReDim d(1 To n - 1)

    For i = 1 To n
        T(i) = Range("A8").Offset(i, 0)
        rzero(i) = Range("B8").Offset(i, 0)
    Next i

    Call CubicSpline(n, T, rzero, a, b, c, d)

    Dim npoint As Integer: npoint = Range("npoint").Value
    Dim term As Double

    Range("A25:B200").ClearContents

    k = 0
    For i = 1 To n - 1
        For j = 0 To npoint
            k = k + 1
            term = T(i) + j * (T(i + 1) - T(i)) / (npoint + 1)
            Range("A24").Offset(k, 0) = term
            Range("B24").Offset(k, 0) = a(i) + b(i) * term + c(i) * term ^ 2 + d(i) * term ^ 3
        Next j
    Next i

    Range("A24").Offset(k + 1, 0) = T(n)
    Range("B24").Offset(k + 1, 0) = rzero(n)

End Sub
```

Code 6.7: VBA code of the GenYieldCurve( ) routine.