13. Spline Interpolations - Concepts and Examples

Cheng Peng

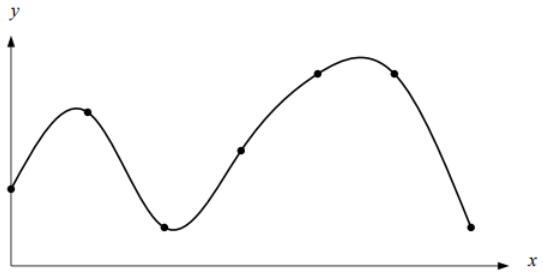
West Chester University

Contents

1 Introduction

We will start with the concepts of various basic concepts of spline curves before introducing the cubic smoothing spline for estimating the functions with given points (knots).

Roughly speaking, splines are functions that are piece-wise polynomials. The coefficients of the polynomial differ from interval to interval, but the order of the polynomial is the same. An essential feature of splines is that function is continuous - i.e. has no breaks on the boundaries between two adjacent intervals. That is, they create smooth curves out of irregular data points.



- Suppose that $n+1$ distinct points $x_0, x_1, \cdots, x_n$ have been specified and satisfy $x_0 < x_1 < \cdots < x_n$. *These points are called knots.*
- Suppose also that an integer $k \geq 0$ has been prescribed. A spline function of degree k having knots $x_0, x_1, \cdots, x_n$ is a function $S(\cdot)$ such that:
  - On each interval $[x_{i-1}, x_i]$, $S(\cdot)$ is a polynomial of degree $\leq k$.

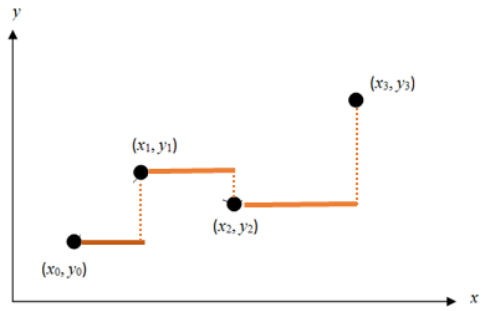
– $S(\cdot)$ has a continuous $(k-1)$-th derivative on $[x_0, x_n]$.

That is if $S(\cdot)$ is a piece-wise polynomial of degree at most 3 having continuous derivatives of all orders up to 2.

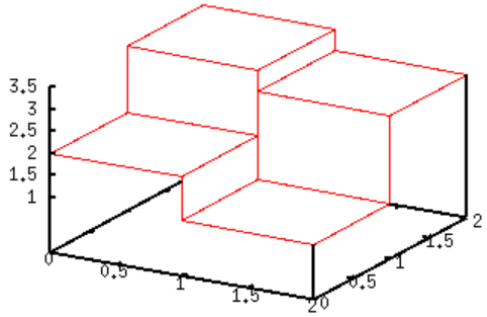
Example 1 Spline of degree 0 is piece-wise constant. A spline of degree 0 can be given explicitly in the form

$$S(x) = \begin{cases} S_0(x) = c_0, & x \in [x_0, x_1] \\ S_1(x) = c_1, & x \in [x_1, x_2] \\ \vdots & \vdots \\ S_{n-1}(x) = c_{n-1}, & x \in [x_{n-1}, x_n] \end{cases}.$$

The intervals $[x_{i-1}, x_i]$ do not intersect each other and so no ambiguity arises in defining such a function at the knots. For example, in the following four-knot data, the zero-degree spline is graphically represented below.



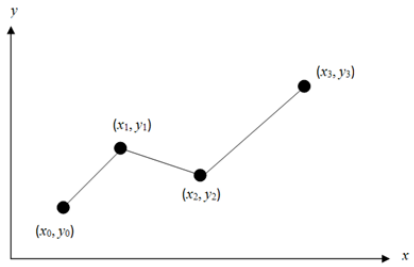
The three dimensional zero-degree spline can be similarly constructed.



We have also used this spline interpolation in Riemann sum in multivatiable integration in calculus. https://demonstrations.wolfram.com/ApproximatingADoubleIntegralWithCuboids/

2 Linear Splines

Linear spline interpolation is simply a line plot that connects all consecutive points $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1}), (x_n, y_n)\}$ $(x_0, x_1, \cdots, x_n)$. So if the above data is given in ascending order, the linear splines are given by $y_i = f(x_i)$.



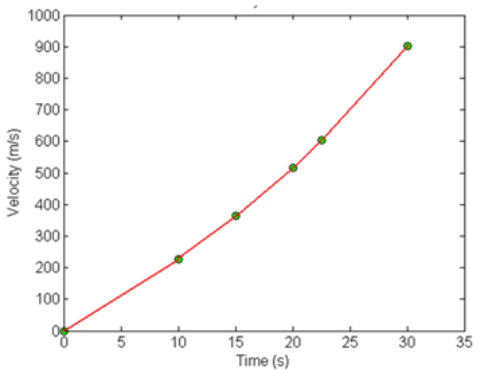
The function of the above curve (i.e., line plot) is given below.

$$f(x) = \begin{cases} f(x_0) + \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0), & x_0 \le x \le x_1 \\ f(x_1) + \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1), & x_1 \le x \le x_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ f(x_{n-1}) + \dfrac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}(x - x_{n-1}), & x_{n-1} \le x \le x_n \end{cases}$$

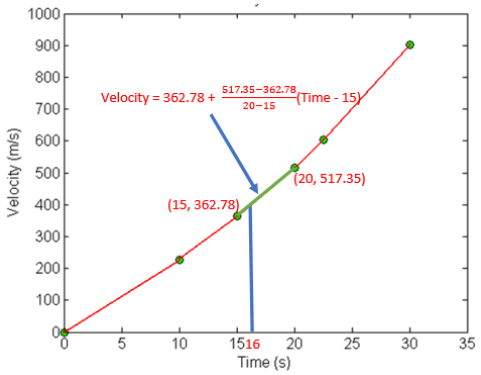
Example 2: The upward velocity of a rocket is given as a function of time in the table below. Using the linear spline to determine the value of the velocity at $t = 16$ seconds.

t (s)	$v(t)$ (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

The linear spline is plotted in the following



We can use the data table to calculate the slope of each individual line segment in the above curve and express the spline function explicitly. The predicted velocity at $Time = 16$ can be found using the piece of the spline in the figure below.



That is,

$$\text{Velocity}_{pred} = 362.78 + \frac{517.35 - 362.78}{20 - 15}(16 - 15) = 393.694.$$

R/MATLAB Program to implement **Linear Spline**.

Assume n given points with coordinates $\{(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)\}$ $(x_0 < x_1 < \cdots < x_n)$. We use the vectorized function `which()` in R or `find()` in MATLAB to avoid using a loop in the algorithm.

```
INPUT: input nodes    (xi,yi) (i = 1, 2, ..., n)
       x.new          (a vector of x values for predicting y.new)

OUTPUT: y.new

STEP 1: Initializing:
        m = number of input x values
        y.new = NULL  (storing pred y)
STEP 2: FOR i = 1 TO m DO;
           IF x[k] <= x.new[i] < x[k+1] DO
               y.new[i] = ((y[k+1]-y[k])/(x[k+1] - x[k]))*(x.new-x[k]) + y[k]
           ENDIF
        ENDFOR
STEP 3: RETURN y.new
```

R Code

```r
LSpline = function(x,         # x-coordinates of the input knots
                   y,         # y-coordinates of the input knots
                   x.new      # the new x values to be evaluated and returned
                   ){
    m = length(x.new)
    y.new = NULL
    for (i in 1:m){
       k = which(x >= x.new[i])[1]
       y.new[i] = ((y[k+1]-y[k])/(x[k+1] - x[k]))*(x.new[i]-x[k]) + y[k]
    }
```

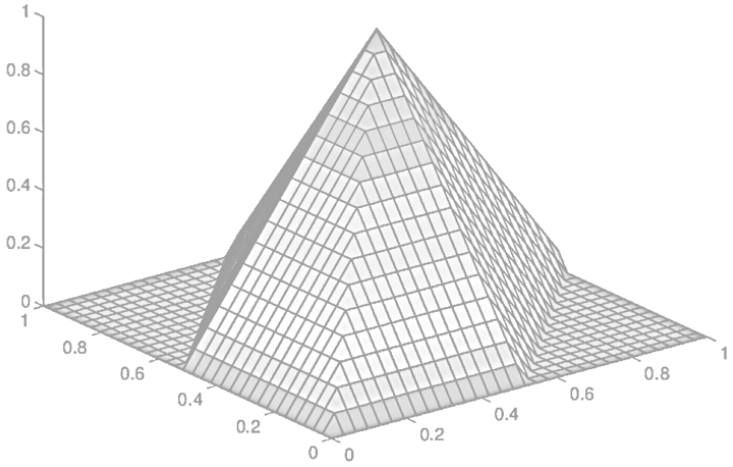
```
    y.new
}
```

Example (reproduce the results of **Examples 2**) using the above function.

```
x = c(0, 10, 15, 20, 22.5, 30)
y = c(0, 227.04, 362.78, 517.35, 602.97, 901.67)
LSpline(x = x, y = y, x.new = c(10, 15, 16, 20))
```

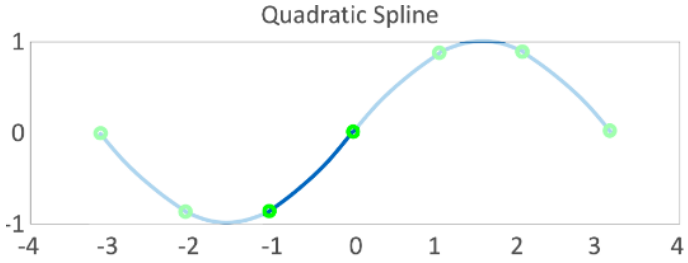
```
## [1] 227.040 362.780 380.358 517.350
```

As expected, linear spline idea can be used in the high dimensional space.



3 Quadratic Spline Interpolation

Unlike linear spline interpolation in which two consecutive knots are connected by a line segment, in quadratic spline interpolation, two consecutive knots are connected by a curve of quadratic function, and every adjacent quadratic curve is connected smoothly (i.e., the derivative of the resulting quadratic spline exists at all **inner knots**).



3.1 Formulation of Quadratic Spline

In these splines, a quadratic polynomial approximates the data between two consecutive data points. Given $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1}), (x_n, y_n), \}$, fit quadratic splines through the data. The splines are given by

$$f(x) = \begin{cases} a_1x^2 + b_1x + c_1, & x_0 \le x \le x_1 \\ a_2x^2 + b_2x + c_2, & x_1 \le x \le x_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_nx^2 + b_nx + c_n, & x_{n-1} \le x \le x_n \end{cases}$$

There are $3n$ coefficients (a_i, b_i, c_i) for $i = 1, 2, \cdots, n$ that must satisfy the following conditions so that the adjacent quadratic curves are connected smoothly.

- **Adjacent connection** implies that the quadratic curves pass through every knot. We have the following $2n$ equations.

$$a_1{x_0}^2 + b_1x_0 + c_1 = f(x_0) \quad \text{Knot 1}$$

$$\left.\begin{aligned} a_1{x_1}^2 + b_1x_1 + c_1 = f(x_1) \\ a_2{x_1}^2 + b_2x_1 + c_2 = f(x_1) \end{aligned}\right\} \quad \text{Knot 2}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\left.\begin{aligned} a_{i-1}{x_{i-1}}^2 + b_{i-1}x_{i-1} + c_{i-1} = f(x_{i-1}) \\ a_i{x_{i-1}}^2 + b_ix_{i-1} + c_i = f(x_{i-1}) \end{aligned}\right\} \quad \text{Knot } i-1$$

$$\left.\begin{aligned} a_i{x_i}^2 + b_ix_i + c_i = f(x_i) \\ a_{i+1}{x_i}^2 + b_{i+1}x_i + c_{i+1} = f(x_i) \end{aligned}\right\} \quad \text{Knot } i$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$\left.\begin{aligned} a_{n-1}{x_{n-1}}^2 + b_{n-1}x_{n-1} + c_{n-1} = f(x_{n-1}) \\ a_n{x_{n-1}}^2 + b_nx_{n-1} + c_n = f(x_{n-1}) \end{aligned}\right\} \quad \text{Knot } n-1$$

$$a_n{x_n}^2 + b_nx_n + c_n = f(x_n) \quad \text{Knot n}$$

- **Smooth connection** requires the first-order derivatives of adjacent quadratic curves at each knot to be equal. This gives the following $n - 1$ equations:

$$2a_kx_k + b_k = 2a_{k+1}x_k + b_{k+1}$$

for $k = 1, 2, \cdots, n - 1$. that can be written as

$$(2x_k)a_k + b_k - (2x_k)a_{k+1} - b_{k+1} = 0$$

- **Default Assumption**. A common assumption we can take is to set $a_1 = 0$ or $a_1 = a_n$.

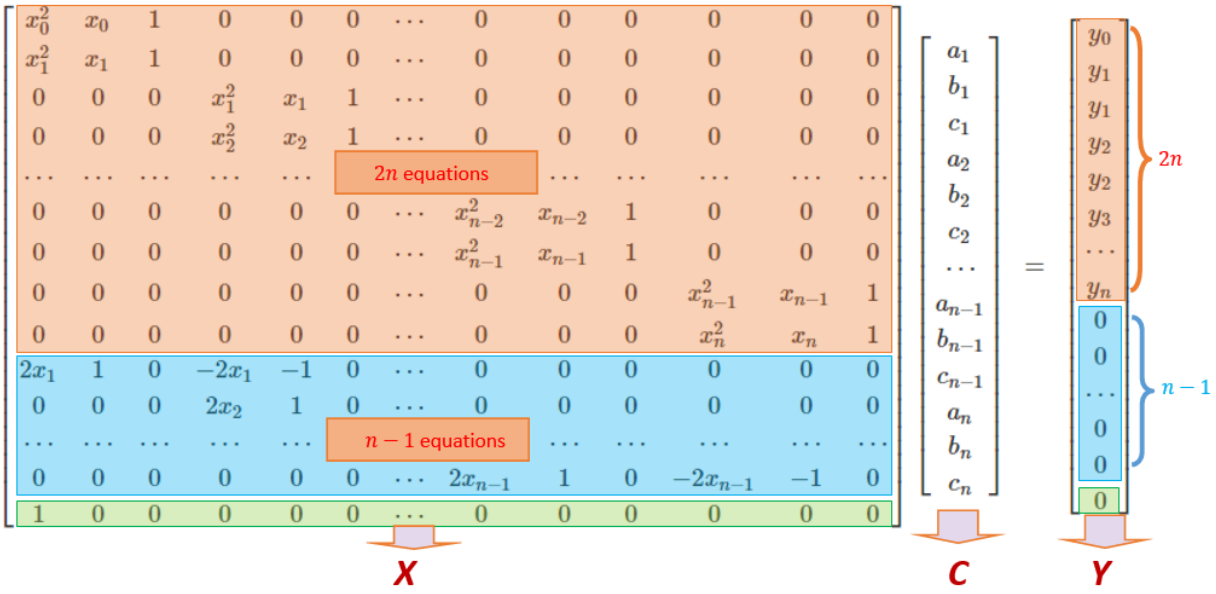
For given $n+1$ points ($n-1$ interior points), we can solve for the $3n$ coefficients for the n quadratic equations.

3.2 Matrix Representation of Quadratic Spline Problem

Since the knots will be given, the actual unknowns are the coefficients of the quadratic functions. We can write the spline problem in the following matrix form.

$$\begin{bmatrix}
x_0^2 & x_0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
x_1^2 & x_1 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & x_1^2 & x_1 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2^2 & x_2 & 1 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & x_{n-2}^2 & x_{n-2} & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & x_{n-1}^2 & x_{n-1} & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & x_{n-1}^2 & x_{n-1} & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & x_n^2 & x_n & 1 \\
2x_1 & 1 & 0 & -2x_1 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2x_2 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 2x_{n-1} & 1 & 0 & -2x_{n-1} & -1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ \cdots \\ a_{n-1} \\ b_{n-1} \\ c_{n-1} \\ a_n \\ b_n \\ c_n \end{bmatrix}
=
\begin{bmatrix} y_0 \\ y_1 \\ y_1 \\ y_2 \\ y_2 \\ y_3 \\ \cdots \\ y_n \\ 0 \\ 0 \\ \cdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The annotated version of the above matrix equation is given below



3.3 Algorithm of Quadratic Spline

The above matrix representation of the quadratic spline problem is a linear system that has a unique solution. Using the algorithm we developed earlier to solve the linear system:

$$XC = Y$$

The solution is

$$C = X^{-1}Y$$

The following pseudo-code will be based on this solution based on given knots $\{(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)\}$.

Q-Spline Algorithm

```
INPUT:   (n+1) knots    (vertical and horizontal coordinates)
         x.new          (to be used to evaluate the spline function)
OUTPUT:  y.new

STEP 1:  Define X and Y
         m = number of values in x.new
         y.new = NULL    (to store values of Q-spline function at x.new)
STEP 2:  solve for C from XC = Y       (coefficients  of Q-spline polynomials)
STEP 3:  FOR i = 1 TO m DO:
            IF x[k] <= x.new[i] < x[k+1] DO:
               y.new[i] = C[3k]*x.new[i]^2 + C[3k+1]*x.new[i] + C[3k+2]
            ENDIF
         ENDFOR
STEP 4: RETURN y.new and coefficients if needed.
```

Next, we write an R function based on the above pseudo-code. We use vector operations in the code to reduce loops.

```r
QSpline = function(x,         # x-coordinates of the input knots
                   y,         # y-coordinates of the input knots
                   x.new      # the new x values to be evaluated and returned
              ){
    m = length(x.new)
    n = length(x)-1
    y.new = NULL
    ##
    Y = rep(0, 3*n)
    Y[1] = y[1]
    ##
    A1 = matrix(0, ncol = 3*n, nrow = 2*n)      # continuity condition
    A2 = matrix(0, ncol = 3*n, nrow = n -1)     # smooth condition
    A3 = matrix(0, ncol = 3*n, nrow = 1)        # default initial condition
    for (i in 1:n){  # pay attention to the indexes
      A1[2*i-1,(3*(i-1)+1):(3*(i-1)+3)] = c(x[i]^2, x[i], 1)
      A1[2*i,(3*(i-1)+1):(3*(i-1)+3)] = c(x[i+1]^2, x[i+1], 1)
      if(i == n) break
      A2[i, (3*(i-1)+1):(3*(i-1)+6)] = c(2*x[i+1], 1, 0, -2*x[i+1], -1, 0)
      ##
```

```
     }
    A3[1,1] = 1
    X = rbind(A1, A2, A3)
    ##
    Y = rep(0, 3*n)
    Y[1] = y[1]
    for (i in 2:n){
        Y[2*i-2] = y[i]
        Y[2*i-1] = y[i]
       }
    Y[2*n] = y[n+1]
    C=solve(X)%*%Y
    a = C[seq(1,3*n, by = 3),]
    b = C[seq(2,3*n, by = 3),]
    c = C[seq(3,3*n, by = 3),]
    coef = round(cbind(a =a, b = b, c = c),3)
    ## Prediction
    for (j in 1:m){
      k = which(x >= x.new[j])[1]
      y.new[j] = a[k]*x.new[j]^2 + b[k]*x.new[j] + c[k]
    }
   list(y.new = y.new, QSpoly.coef = coef, QSeq.Matrix = X, Y = Y)
}
```

Example 3: (**Velocity example continued**) The upward velocity of a rocket is given as a function of time in the table below. Using the linear spline to determine the value of the velocity at $t = 16$ seconds.

t (s)	$v(t)$ (m/s)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

We fit the data with a quadratic spline with the assumption that $a_1 = 0$. The matrix form of the final system of $3 \times (6 - 1) = 15$ linear equations are given below.

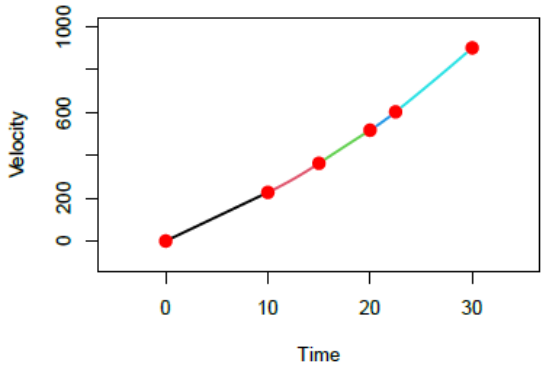
$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
100 & 10 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 100 & 10 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 225 & 15 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 225 & 15 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 400 & 20 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 400 & 20 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 506.25 & 22.5 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 506.25 & 22.5 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 900 & 30 & 1 \\
20 & 1 & 0 & -20 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 30 & 1 & 0 & -30 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 40 & 1 & 0 & -40 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 1 & 0 & -45 & -1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \\ a_5 \\ b_5 \\ c_5 \end{bmatrix}
=
\begin{bmatrix} 0 \\ 227.04 \\ 227.04 \\ 362.78 \\ 362.78 \\ 517.35 \\ 517.35 \\ 602.97 \\ 602.97 \\ 901.67 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The last row in the argument matrix reflects the default assumption $a_1 = 0$. Solve the above system of

equations using a computer program, we have the solutions summarized in the following table.

i	a_i	b_i	c_i
1	0	22.704	0
2	0.8888	4.928	88.88
3	−0.1356	35.66	−141.61
4	1.6048	−33.956	554.55
5	0.20889	28.86	−152.13

Using the above results, we plot the quadratic spline with the following R code.



Using the above R function, we have the above results.

```
x = c(0, 10, 15, 20, 22.5, 30)
y = c(0, 227.04, 362.78, 517.35, 602.97, 901.67)
x.new = c(0, 10, 15, 16, 20)
QSpline(x = x, y = y, x.new = x.new)
```

```
## $y.new
## [1]   0.0000 227.0400 362.7800 422.0828 517.3500
##
## $QSpoly.coef
##           a       b       c
## [1,]  0.000  22.704    0.00
## [2,]  0.889   4.928   88.88
## [3,] -0.136  35.660 -141.61
## [4,]  1.605 -33.956  554.55
## [5,]  0.209  28.860 -152.13
##
## $QSeq.Matrix
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]  [,10] [,11] [,12]  [,13]
##  [1,]    0    0    1    0    0    0    0    0    0   0.00   0.0     0   0.00
##  [2,]  100   10    1    0    0    0    0    0    0   0.00   0.0     0   0.00
##  [3,]    0    0    0  100   10    1    0    0    0   0.00   0.0     0   0.00
##  [4,]    0    0    0  225   15    1    0    0    0   0.00   0.0     0   0.00
##  [5,]    0    0    0    0    0    0  225   15    1   0.00   0.0     0   0.00
##  [6,]    0    0    0    0    0    0  400   20    1   0.00   0.0     0   0.00
##  [7,]    0    0    0    0    0    0    0    0    0 400.00  20.0     1   0.00
##  [8,]    0    0    0    0    0    0    0    0    0 506.25  22.5     1   0.00
##  [9,]    0    0    0    0    0    0    0    0    0   0.00   0.0     0 506.25
## [10,]    0    0    0    0    0    0    0    0    0   0.00   0.0     0 900.00
## [11,]   20    1    0  -20   -1    0    0    0    0   0.00   0.0     0   0.00
## [12,]    0    0    0   30    1    0  -30   -1    0   0.00   0.0     0   0.00
```

```
## [13,]    0    0    0    0    0    0   40    1    0 -40.00  -1.0     0   0.00
## [14,]    0    0    0    0    0    0    0    0    0  45.00   1.0     0 -45.00
## [15,]    1    0    0    0    0    0    0    0    0   0.00   0.0     0   0.00
##       [,14] [,15]
##  [1,]   0.0     0
##  [2,]   0.0     0
##  [3,]   0.0     0
##  [4,]   0.0     0
##  [5,]   0.0     0
##  [6,]   0.0     0
##  [7,]   0.0     0
##  [8,]   0.0     0
##  [9,]  22.5     1
## [10,]  30.0     1
## [11,]   0.0     0
## [12,]   0.0     0
## [13,]   0.0     0
## [14,]  -1.0     0
## [15,]   0.0     0
##
## $Y
##  [1]   0.00 227.04 227.04 362.78 362.78 517.35 517.35 602.97 602.97 901.67
## [11]   0.00   0.00   0.00   0.00   0.00
```