CAGD - Homework 3

Josefine Stål & Erik Ackzell

September 26, 2016

Task 1

As we understood the task we want to give all the basis functions for the two knot sequences $U_1 = \{0, 0, 1, 1\}$ and $U_2 = \{0, 0, 0, 1, 1, 1\}$. We use the recursion formula for calculating the basis functions for each index i,

$$N_i^0(u) = \begin{cases} 1, & \text{if } u \in [u_i, u_{i+1}) \\ 0, & \text{else} \end{cases}$$

$$N_i^n(u) = \frac{u - u_i}{u_i + n - u_i} N_i^{n-1} + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n+1}(u)$$

where u_i are the knots. We can easily calculate how many basis functions there are for each degree by the relation

$$\#\text{knots} - 1 = \text{degree} + \#\text{basis functions}$$

The basis functions for $U1 = \{0, 0, 1, 1\}$ are the following:

$$N_0^1(u) = (1 - u)N_1^0(u)$$
$$N_1^1(u) = uN_1^0(u)$$
$$N_0^2(u) = 2u(1 - u)N_1^0(u)$$

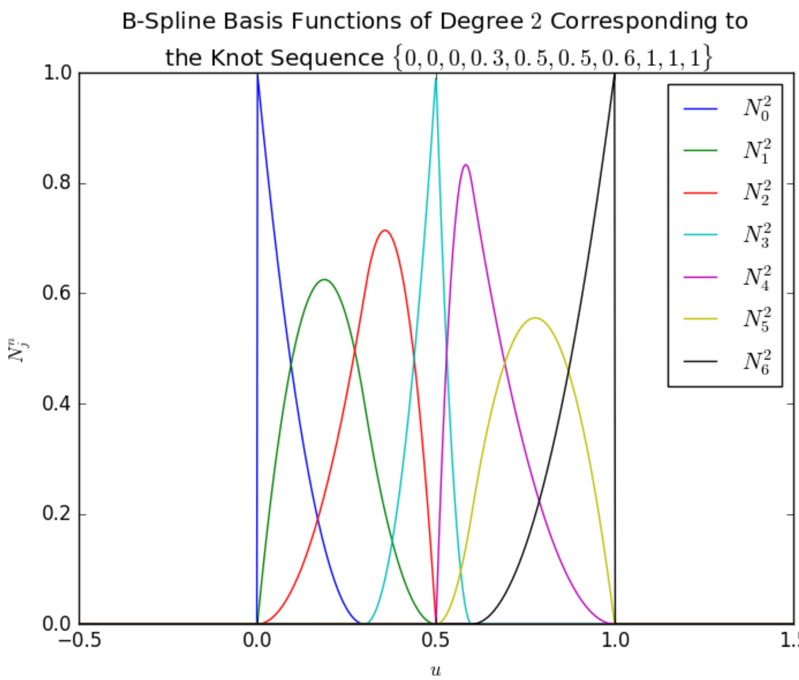
The basis functions for $U2 = \{0, 0, 0, 1, 1, 1\}$ are as follows:

$$N_0^1(u) = 0$$
$$N_1^1(u) = (1 - u)N_2^0(u)$$
$$N_2^1(u) = uN_2^0(u)$$
$$N_3^1(u) = 0$$
$$N_0^2(u) = (1 - u)^2 N_2^0(u)$$
$$N_1^2(u) = 2u(1 - u)N_2^0(u)$$
$$N_2^2(u) = u^2 N_2^0(u)$$
$$N_0^3(u) = 3u(1 - u)^2 N_2^0(u)$$
$$N_1^3(u) = 3u^2(1 - u)N_2^0(u)$$
$$N_0^4(u) = 6u^2(1 - u)^2 N_2^0(u)$$

As can be seen by the basis functions above we have a clear pattern in the coefficents to and the multiples of u and $(1 - u)$ respectively.

Task 2

The algorithm that calculates and plots the values of the basis functions corresponding to any given knot sequence and degree, can be found in Appendix I. In this task the degree is 2 and the knot sequence $\{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$.



Task 3

In this task we implement a method to check whether a real number is in an interval with full support, given a knot sequence, a degree of the spline basis and the number itself. The code can be seen in Appendix I.
We tried our code for three different knot sequences, $k_1 = (0, 0, 1, 1)$, $k_2 = (0, 0, 0, 1, 1, 1)$ and $k_3 = (0, 0, 0, 0.3, 0.5, 0.6, 1, 1, 1)$, all with quadratic splines. For each of the scenarios, we tested our code with the real numbers $\{0.12, 0.24, 0.4, 0.53, 0.78, 0.8\}$.

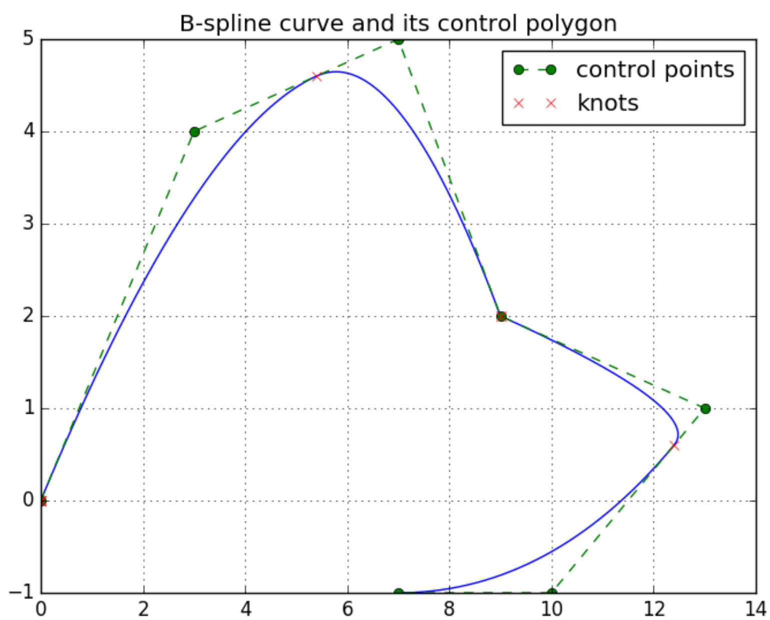
- For k_1, we found that all points were in intervals **without** full support.
- For k_2, we found that all points were in intervals **with** full support.
- For k_3, we found that all points were in intervals **with** full support.

The results were as expected, as an interval has full support when using quadratic splines, if there are 3 or more knots to the left of the left point of the interval as well as to the right of the right point of the interval.

Task 4

In this task we plot the spline curve with knots $\{0, 0, 0, 0.3, 0.5, 0.5, 0.6, 1, 1, 1\}$ and control points $\{(0, 0), (3, 4), (7, 5), (9, 2), (13, 1), (10, -1), (7, -1)\}$. We do this by using the definition of the spline, $C(u) = \sum_{i=0}^{m} d_i N_i^n(u)$, where d_i are the control points and N_i^n are the spline basis functions. The methods implemented in the previous tasks are used to achieve this.

The plot can be seen below and the code can be seen in Appendix I. As the number of knots is 10 and the number of control points is 7, the degree of the curve is $10 - 1 - 7 = 2$. Thus, the kink at the knot 0.5 is expected, as the curve is only $2 - 2 = 0$ times continuous at a knot of multiplicity 2.



Appendix I

```
import scipy
import pylab
import numpy as np


class bspline:

    def __init__(self, knots, degree=None, controlpoints=None):
        self.knots = knots

        if controlpoints is not None:
            self.controlpoints = controlpoints
            self.degree = len(self.knots) - 1 - len(self.controlpoints)
            if degree:
                assert degree == self.degree, \
                       'Given degree is wrong. Check the knots and
                           control' + \
                       'points or do not define a degree yourself.'
        else:
            self.degree = degree

    def __call__(self, u):
        """
        Evaluates the spline at a point u, using the spline definition.
        """
        S = sum([self.controlpoints[i] *
             self.get_basisfunc(k=self.degree,j=i)(u)
                 for i in range(len(self.controlpoints))])
        return S

    def has_full_support(self, u):
        """
        This method checks if the point u is in an interval with full
            support.
        """
        if min(scipy.count_nonzero(self.knots < u),
               scipy.count_nonzero(self.knots > u)) > self.degree:
            return True
        else:
            return False

    def get_basisfunc(self, k, j):
        """
        Method that returns a function which evaluates the basis
            function of
        degree k with index j at point u.
```

```
    """
    def basisfunction(u, k=k, j=j):
        """
        Method to evaluate the the basis function N^k with index j at
        point u.
        u (float): the point where to evaluate the basis function
        k (int): the degree of the basis function
        j (int): the index of the basis function we want to evaluate
        knots (array): knot sequence u_i, where i=0,...,K
        """
        if k == 0:
            return 1 if self.knots[j] <= u < self.knots[j+1] \
                    else 0
        else:
            try:
                a0 = 0 if self.knots[j+k] == self.knots[j] \
                      else (u - self.knots[j]) / (self.knots[j+k] -
                                                  self.knots[j])
                a1 = 0 if self.knots[j+k+1] == self.knots[j+1] \
                      else (self.knots[j+k+1] - u) /
                          (self.knots[j+k+1] -
                                                      self.knots[j+1])
                basisfunc = a0 * basisfunction(u, k=k-1, j=j) + \
                           a1 * basisfunction(u, k=k-1, j=j+1)
            except IndexError:
                numBasisfunc = len(self.knots) - 1 - k
                raise IndexError('Invalid index. There are no more
                    than {} basis functions for the given problem,
                    choose an ' \
                          'index lower than the number of basis
                              functions.'.format(numBasisfunc))

            return basisfunc
    return basisfunction

def plot(self):
    """
    This method plots the spline.
    """
    ulist = scipy.linspace(self.knots[0], self.knots[-1], 1000)
    ulist = [u for u in ulist if self.has_full_support(u=u)]
    pylab.plot(*zip(*[self(u=u) for u in ulist]))
    pylab.plot(*zip(*self.controlpoints), 'o--', label='control
        points')
    pylab.plot(*zip(*[self(u=u) for u in self.knots]), 'rx',
        label='knots')
    pylab.legend()
    pylab.grid()
    pylab.title('B-spline curve and its control polygon')
    pylab.show()
```