Definite Integration of Polynomials with Polynomials as Limits

Robin Schmidt (www.rs-met.com)

November 13, 2011

We consider the problem of finding the coefficients $q_0, \ldots, q_{N_q}$ of an N_qth order polynomial $q(x)$ that is given by a definite integral over another polynomial $p(u)$ where the lower and upper integration limits are also given by polynomials $a(x)$ and $b(x)$, such that:

$$q(x) = \int_{a(x)}^{b(x)} p(u)du \tag{1}$$

The orders of the polynomials $p(u), a(x), b(x)$ are denoted as N_p, N_a and N_b, respectively.

The Antiderivative of $p(u)$

We assume that our integrand polynomial $p(u)$ is represented by an array of its coefficients $p_0, \ldots, p_{N_p}$, so we have:

$$p(u) = \sum_{i=0}^{N_p} p_i u^i \tag{2}$$

We will denote this coefficient array in vector notation as $\mathbf{p} = [p_0, \ldots, p_{N_p}]^T$. The dimensionality of this vector is $N_p + 1$. We will denote the antiderivative of $p(u)$ as $P(u)$. This antiderivative, in terms of its polynomial coefficients is given by:

$$P(u) = \sum_{i=0}^{N_P} P_i u^i = \int p(u)du = P_0 + \sum_{i=1}^{N_P} \frac{p_{i-1}}{i} u^i \tag{3}$$

The order N_P of this antiderivative $P(u)$ is one higher than that of $p(u)$, so we have: $N_P = N_p + 1$. The coefficient P_0 for the constant term u^0 may be chosen arbitrarily - it represents our integration constant. As we have no specific reason to do otherwise, we will choose $P_0 = 0$. The coefficient vector of $P(x)$ is denoted as: $\mathbf{P} = [P_0, \ldots, P_{N_P}]^T$ and its elements can be computed from the elements in the coefficient vector $\mathbf{p}$ via:

$$P_i = \begin{cases} 0 & \text{for } i = 0 \\ \frac{p_{i-1}}{i} & \text{for } i = 1, \ldots, N_P \end{cases} \tag{4}$$

In pseudo MatLab/Octave code, the function to find the coefficient-array of the antiderivate P of a polynomial p, also represented as coefficient-array, could look like this:

```
function P = integratePolynomial(p, P0);
if( nargin < 2 )
 P0 = 0;
end
P = [P0, p ./ (1:length(p))];
```

Inserting the Integration Limits

Having found a means to construct the antiderivative of $p(x)$, we may go back to (1) and apply the the fundamental theorem of calculus $\int_a^b f(x)dx = F(b) - F(a)$:

$$q(x) = \int_{a(x)}^{b(x)} p(u)du = P(b(x)) - P(a(x)) \tag{5}$$

The term $P(b(x))$ represents a composition (or nesting) of the two polynomials P and b, that is: for an input value x, we first evaluate $y = b(x)$ and then evaluate $P(y)$. We shall denote this composed polynomial as $B(x)$. The same reasoning applies to $P(a(x))$. So, the formula above suggests the following algorithm:

```
function q = defPolyIntWithPolyLims(p, a, b);
P = integratePolynomial(p);
B = composePolynomials(b, P);
A = composePolynomials(a, P);
q = subtractPolynomials(B, A);
```

where we assume that we have functions at our disposal that perform integration, composition and subtraction of polynomials. All these functions expect polynomial coefficient vectors as inputs and return another polynomial coefficient vector as result. The implementation of the `integratePolynomial` function was given in the previous section, `composePolynomials` and `subtractPolynomials` will be given in subsequent sections.

Composition of Polynomials

In this section, we consider the problem of finding a polynomial $c(x)$ that is a composition of two polynomials $a(x)$ and $b(x)$ which have orders N_a and N_b respectively. The order of the resulting polynomial $c(x)$ will come out as $N_c = N_a N_b$. We write the polynomials $a(x), b(x), c(x)$ as:

$$a(x) = \sum_{i=0}^{N_a} a_i x^i, \qquad b(x) = \sum_{j=0}^{N_b} b_j x^j, \qquad c(x) = \sum_{k=0}^{N_c} c_k x^k \tag{6}$$

where $c(x)$ is taken to be the composition of $a(x)$ and $b(x)$:

$$c(x) = b\left(a(x)\right) = \sum_{j=0}^{N_b} b_j \left(\sum_{i=0}^{N_a} a_i x^i \right)^j \tag{7}$$

and our goal is to find an algorithm to compute the array of coefficients $c_0, \ldots, c_{N_c}$ from the two coefficient arrays $a_0, \ldots, a_{N_a}$ and $b_0, \ldots, b_{N_b}$. By inspecting equation (7), we recognize that we need coefficient-arrays of successive powers of the polynomial $a(x)$, weight those arrays by a coefficient b_j and accumulate them into our **c**-array. We start from the 0th power and go up to to the N_bth power of $a(x)$. The 0th power is simply the constant 1, the 1st power is the polynomial $a(x)$ itself. The second power is where it becomes interesting: the coefficient-array array that represents the polynomial $(a(x))^2 = (\sum_{i=0}^{N_a} a_i x^i)^2$ can be found by convolving the **a**-array with itself because a multiplication of two polynomials, represented as coefficient arrays, is done by a convolution of these coefficient-arrays. For the next power $(a(x))^3 = (\sum_{i=0}^{N_a} a_i x^i)^3$ we take the result of the previous power, $(a(x))^2$, and convolve it with the **a**-array again. Thus, our overall algorithm should repeatedly convolve the coefficient array of the inner polynomial with itself, weight this result by a coefficient from the outer polynomial and accumulate it into the coefficient-array of the result. In code, this could look like:

```
function c = composePolynomials(a, b);
c    = zeros(1, (length(a)-1)*(length(b)-1)+1);
c(1) = b(1);
an   = 1;
for n=2:length(b)
 an = conv(an, a); % coeffs of a^(n-1)
 c  = c + b(n) * [an, zeros(1, length(c)-length(an))];
end
```

Subtraction of Polynomials

Actually, this is easy: we just subtract the coefficient arrays element-wise. However, some care must be taken, if the polynomials to be subtracted are of different order (i.e. their coefficient arrays are of different length): we will have to zero-pad the shorter coefficient vector for the higher order coefficients appropriately. For generality, we write a function to form a weighted sum and define the subtraction as special case thereof:

```
function r = weightedSumOfPolynomials(p, wp, q, wq);
Lp = length(p);
Lq = length(q);
if( Lp > Lq )
 d = Lp - Lq;
 q = [q, zeros(1, d)];
elseif( Lq > Lp )
 d = Lq - Lp;
 p = [p, zeros(1, d)];
end
r = wp*p + wq*q;

function r = subtractPolynomials(p, q);
r = weightedSumOfPolynomials(p, 1, q, -1);
```