Polynomials

Jordi Cortadella
Department of Computer Science

Outline

• How to represent a polynomial?
• Evaluation of a polynomial.
• Basic operations: sum, product and division.
• Greatest common divisor.

Representation of polynomials

\[ P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \]

<table>
<thead>
<tr>
<th>(a_n)</th>
<th>(a_{n-1})</th>
<th>(\cdots)</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

typedef vector<double> Polynomial;

Properties of the representation:
• \(a_n \neq 0\) (the topmost element is not zero)
• \(P(x) = 0\) is represented with an empty vector.

Polynomial evaluation (Horner’s scheme)

• Design a function that evaluates the value of a polynomial.
• A polynomial of degree \(n\) can be efficiently evaluated using Horner’s algorithm:

\[ P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \]

\[ (\cdots ((a_n x + a_{n-1}) x + a_{n-2}) x + \cdots) x + a_0 \]

• Example:

\[ 3x^3 - 2x^2 + x - 4 = ((3x - 2)x + 1)x - 4 \]
Polynomial evaluation (Horner’s scheme)

```c++
double PolyEval(const Polynomial& P, double x) {
    double eval = 0;
    /* Invariant: the polynomial has been evaluated up to the coefficient i+1 using Horner’s scheme */
    for (int i = P.size()-1; i >= 0; --i) {
        eval = eval*x + P[i];
    }
    return eval;
}
```

Product of polynomials

```c++
Polynomial PolyProduct(const Polynomial& P, const Polynomial& Q) {
    // Special case for a polynomial of size 0
    if (P.size()*Q.size() == 0) return Polynomial(0);
    Polynomial R(P.size() + Q.size() - 1, 0);
    for (int i = 0; i < P.size(); ++i) {
        for (int j = 0; j < Q.size(); ++j) {
            R[i + j] += P[i]*Q[j];
        }
    }
    return R;
}
```

**Example:**

\[ P(x) = 2x^3 + x^2 - 4 \]
\[ Q(x) = x^2 - 2x + 3 \]
\[ (P \cdot Q)(x) = 2x^5 + (-4 + 1)x^4 + (6 - 2)x^3 + 8x - 12 \]
\[ (P \cdot Q)(x) = 2x^5 - 3x^4 + 4x^3 + 8x - 12 \]
Sum of polynomials

- Note that over the real numbers, \( \deg(P \times Q) = \deg(P) + \deg(Q) \) (except if \( P = 0 \) or \( Q = 0 \)).

So we know the size of the result vector a priori.

- This is not true for the polynomial sum, e.g.

\[ \deg((x + 5) + (-x - 1)) = 0 \]

A function to normalize a polynomial might be useful in some algorithms, i.e., remove the leading zeros to guarantee the most significant coefficient is not zero.

```cpp
// Resizes the polynomial to guarantee that the leading coefficient is not zero.
void PolyNormalize(Polynomial& P) {
    while (P.size() > 0 and P[P.size()-1] == 0) P.pop_back();
}
```

\[ \text{Polynomial PolySum(const Polynomial& P, const Polynomial& Q) { }
\]
\[
\text{    int maxsize = max(P.size(), Q.size());}
\]
\[
\text{    Polynomial R;}
\]
\[
\text{    // Inv: R[0..i-1] = (P+Q)[0..i-1]}
\]
\[
\text{    for (int i = 0; i < maxsize; ++i) {
\text{        double s;}
\text{        if (i >= P.size()) s = Q[i];}
\text{        else if (i >= Q.size()) s = P[i];}
\text{        else s = P[i] + Q[i];}
\text{        R.push_back(s);}
\text{    }}
\]
\[
\text{    // Remove zeros from the leading locations}
\text{    PolyNormalize(R);
\text{    return R;}
\text{}}
\]

Euclidean division of polynomials

- For every pair of polynomials \( A \) and \( B \), such that \( B \neq 0 \), find \( Q \) and \( R \) such that

\[ A = B \cdot Q + R \]

and \( \deg(R) < \deg(B) \).

\( Q \) and \( R \) are the only polynomials satisfying this property.

- Classical algorithm: \textit{Polynomial long division}. 
Polynomial long division

// Calculates de quotient (Q) and remainder (R)
// of the Euclidean division A/B. The result is
// returned through the parameters Q and R.

void PolyDivision(const Polynomial& A, const Polynomial& B, Polynomial& Q, Polynomial& R);

A: 2x^5 + 6x^4 - 3x^3 + x^2 - 2  
B: -x^3 + x^2  
Q: (2x^3 + x - 1)  
R: (x^2 + 3x - 2)

Invariant: A = B·Q + R

R: 2 6 -3 1 0 -1  
B: 2 0 1 -1  
Q: 6 -4 2 0 -1  
-4 -1 3 -1  
-1 5 -3

Polynomial long division

void PolyDivision(const Polynomial& A, const Polynomial& B, Polynomial& Q, Polynomial& R) {
    R = A;
    if (A.size() < B.size()) { // Special case: degree(A) < degree(B)
        Q = Polynomial(0); return;
    }
    Q = Polynomial(A.size() - B.size() + 1, 0);
    int iR = R.size() - 1; // index of the leading coef. of R

    // Every iteration performs a step of the division
    // Invariant: A = B·Q + R
    for (int iQ = Q.size() - 1; iQ >= 0; --iQ) {
        Q[iQ] = R[iR] / B[B.size() - 1]; // new coef. of Q
        R[iR] = 0; // Guarantees an "exact" zero at the leading coef.
        for (int k = B.size() - 2; k >= 0; --k) R[k+iQ] -= Q[iQ]*B[k];
        --iR; // Index to the next leading coef. of Q
    }
    PolyNormalize(R);
}

GCD of two polynomials

Example:

P(x) = x^4 - 7x^2 - 6x
     = x(x + 1)(x + 2)(x - 3)

Q(x) = x^3 - x^2 - 5x - 3
     = (x + 1)^2(x - 3)
gcd(P, Q)(x) = x^2 - 2x - 3
              = (x + 1)(x - 3)
Re-visiting Euclidean algorithm for gcd

// gcd(a, 0) = a
// gcd(a, b) = gcd(b, a%b)

// Returns gcd(a, b)
int gcd(int a, int b) {
    while (b > 0) {
        int r = a%b;
        a = b;
        b = r;
    }
    return a;
}

For polynomials:
- a and b are polynomials.
- a%b is the remainder of the Euclidean division.

Example: \(A(x) = x^4 - 7x^2 - 6x\), \(B(x) = x^3 - x^2 - 5x - 3\).

\[
\begin{array}{|c|c|c|c|}
\hline
A & B & Q & R \\
\hline
x^4 - 7x^2 - 6x & x^3 - x^2 - 5x - 3 & x + 1 & -x^2 + 2x + 3 \\
x^3 - x^2 - 5x - 3 & -x^2 + 2x + 3 & -x - 1 & 0 \\
-x^2 + 2x + 3 & 0 & & \\
\hline
\end{array}
\]

\[x^2 - 2x - 3 = (x + 1)(x - 3)\]

Conclusions
- Polynomials are used very often in numerical analysis.
- Polynomial functions have nice properties: continuous and simple derivatives and antiderivatives.
- There are simple root-finding algorithms to find approximations of the roots.